FOSAD 2001

University Residential Center

Bertinoro, September 16th – 26th

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	2001	16-26 September 2001 Final

4. TITLE AND SUBTITLE
FOSAD 2001. 2nd International School on Foundations of Security and Design Held in Bertinoro, Italy on September 16th-26th 2001.

5. FUNDING NUMBERS
N00014-01-1-10xx

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Bologna
Department of Information Science
Mura Anteo Zamboni 7
I-40127 Bologna, Italy

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research,
European Office
PSC 802 Box 39
FPO AE 09499-0039

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES See: http://www.cs.unibo.it/~aldini/fosad01/index.html
This work relates to Department of the Navy Grant N00014-01-1-10xx issued by the Office of Naval Research International Field Office-Europe. The United States has a royalty-free license throughout the world in all copyrightable material contained herein.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited.

12b. DISTRIBUTION CODE
A

12. ABSTRACT (Maximum 200 words)
Security in computer systems and networks is emerging as one of the most challenging research areas for the future. The main aim of the school is to offer a good spectrum of current research in foundations of security, ranging from programming languages to analysis of protocols, that can be of help for graduate students, young researchers from academia or industry that intend to approach the field. The school covers two weeks (from Monday 17 to Saturday 29, September 2001) and alternates four lecturers per week. Topical papers by the lecturers include:

Validating Firewalls Using Flow Logics by N. Flemming, H.R. Nielson, R.R. Hansen

Revocation and Tracing Schemes for Stateless Receivers by D. Naor, M. Naro, J. Lotspiech

Inividual Authentication in Multiparty Communications by F. Bergadano, D. Cavagnino, B. Crispo

How to Sign Digital Streams by R. Gennaro, P. Rohatgi

Efficient Authentication and Signing of Multicast Streams over Lossy Channels by A. Perrig, R. Canetti, J.D. Tygar, D. Song T.J. Watson

Multicast Security: A Taxonomy and Some Efficient Structures by R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas

Tracing Traitors by B. chor, A. Fiat, M. Naor

An Efficient Public Key Traitor Tracing Scheme by D. Boneh, M. Franklin

Classification of Security Properties by R. Focardi, R. Gorrieri

Access Control: Policies, Models and Mechanisms by P. Samarati, S. De Capitani di Vimercati

Attacking Malicious Code: A Report to the Infosec Research Council by G. McGraw, G. Morrisett

Building Secure Software: Introduction to Software Security by J. Viega, G. McGraw

Software Risk Management for Security by G. McGraw

13. SUBJECT TERMS
ONRIFO, Foreign Reports

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

FOSAD 2001

University Residential Center

Bertinoro, September 16th – 26th

2nd International School On

Foundations Of Security Analysis And Design

FOSAD

17-29 September 2001, Bertinoro, Italy

Application deadline: June 8, 2001
Notification deadline: June 30, 2001
Registration deadline: July 31, 2001

General Information

Security in computer systems and networks is emerging as one of the most challenging research areas for the future. The main aim of the school is to offer a good spectrum of current research in foundations of security, ranging from programming languages to analysis of protocols, that can be of help for graduate students, young researchers from academia or industry that intend to approach the field. As for the previous edition (FOSAD'00), the school covers two weeks (from Monday 17 to Saturday 29, September 2001) and alternates four lecturers per week on monographic courses of about 6/8 hours each. Saturdays are reserved for presentations given by those participants that intend to take advantage of the audience for discussing their current research in the area. The school is organised at the Centro Residenziale Universitario of the University of Bologna, situated in Bertinoro, a small village on a scenic hill with a wonderful panorama, in between Forlì and Cesena (about 50 miles south-east of Bologna, 15 miles to the Adriatic sea).

Sponsorships

Under the auspices of:

- European Association of Theoretical Computer Science - Italian Chapter
- International Federation for Information Processing - IFIP TC1-WG1.7
- European Educational Forum

Sponsors:

- CNR-IAT
- ONR
- Università Ca' Foscari di Venezia - Dipartimento di Informatica
- Università di Bologna

FOSAD 2001

Main Courses Speakers:

- **SCS:** Dominique Bolignano
- **IDS:** Marc Dacier
- **CSP:** Roberto Gorrieri and Riccardo Focardi (slides)
- **SMT:** Rosario Gennaro
- **BSS:** Gary McGraw
- **SAS:** Flemming Nielson

Short Courses Speakers:

S1 Carlo Blundo
S2 Roberto Segala
S3 Iliano Cervesato (slides)
S4 Pierangela Samarati (slides)
S5 Fabio Martinelli
S6 Pino Persiano (slides)

Return to the main page

Timetable

<table>
<tr><td></td><td>16
Sun</td><td>17
Mon</td><td>18
Tue</td><td>19
Wed</td><td>20
Thu</td><td>21
Fri</td><td>22
Sat</td><td>23
Sun</td><td>24
Mon</td><td>25
Tue</td><td>26
Wed</td><td>27
Thu</td><td>28
Fri</td><td>29
Sat</td></tr>
<tr><td>08.00-09.00</td><td rowspan="3">arrivals between 18.00 and 21.00</td><td colspan="12">breakfast</td><td rowspan="4">departures</td></tr>
<tr><td>09.00-11.00</td><td>S1</td><td>SAS</td><td>SAS</td><td></td><td>S4</td><td colspan="3"></td><td>BSS</td><td>BSS</td><td>SCS</td><td>BSS</td><td>SMT</td></tr>
<tr><td>11.00-11.30</td><td colspan="5">coffee break</td><td colspan="3"></td><td colspan="5">coffee break</td></tr>
<tr><td>11.30-13.30</td><td>CSP</td><td>S2</td><td>CSP</td><td>SAS</td><td>S3</td><td colspan="3"></td><td>S6</td><td>IDS</td><td>SMT</td><td>SMT</td><td>IDS</td></tr>
<tr><td>13.30-14.30</td><td rowspan="4">welcome dinner 19.30</td><td colspan="12">lunch</td></tr>
<tr><td>15.00-17.00</td><td>CSP</td><td>S2</td><td>CSP</td><td>SAS</td><td>S4</td><td colspan="3"></td><td>IDS</td><td>SCS</td><td>BSS</td><td>SCS</td><td>SCS</td></tr>
<tr><td>17.00-17.30</td><td colspan="5">tea break</td><td colspan="3"></td><td colspan="3">tea break</td><td rowspan="2">departures</td></tr>
<tr><td>17.30-19.30</td><td colspan="4"></td><td>S3</td><td>S5</td><td colspan="4"></td><td>SMT</td><td>IDS</td></tr>
</table>

SCS: Smart Card Security - Dominique Bolignano (Trusted Logics).

IDS: Intrusion Detection Systems - Marc Dacier (IBM, Zurich Research Laboratory).

SMT: Security for Multimedia Traffic over IP - Rosario Gennaro (IBM, T.J. Watson Research Center).

CSP: Classification of Security Properties - Roberto Gorrieri (Univ. of Bologna) and Riccardo Focardi (Univ. of Venezia).

BSS: Building Secure Software - Gary McGraw (Cigital).

SAS: Static Analysis for Security - Flemming Nielson (Technical Univ. of Denmark, Lyngby).

S1: Introduction to Cryptography I.
S2: Introduction to Cryptography II.
S3: Security Protocol Specification Languages.
S4: Access Control: policies, models, architectures and mechanisms.
S5: Logics for Security.
S6: Security Notions for Public Key Cryptosystems.

Validating Firewalls using Flow Logics

Flemming Nielson, Hanne Riis Nielson, René Rydhof Hansen

Department of Computer Science, Aarhus University,
Ny Munkegade, DK-8000 Aarhus C, Denmark.

E-mail: {fn,hrn,rrh}@daimi.au.dk

Abstract. The ambient calculus is a calculus of computation that allows active processes to communicate and to move between sites. A site is said to be a protective firewall whenever it denies entry to all attackers not possessing the required passwords. We devise a computationally sound test for validating the protectiveness of a proposed firewall and show how to perform the test in polynomial time.

The first step is the definition of a flow logic for analysing the flow of control in mobile ambients; it amounts to a syntax-directed specification of the acceptability of a control flow estimate. The second step is to define a hardest attacker and to determine whether or not there exists a control flow estimate that shows the inability of the hardest attacker to enter; if such an estimate exists, then none of the infinitely many attackers can enter unless they contain at least one of the passwords, and consequently the firewall cannot contain any trap doors.

Keywords: Hardest attackers, static analysis, control-flow analysis, flow logic, mobile ambients, firewalls.

1 Introduction

The ambient calculus is a calculus of computation that is based on traditional process algebras (such as the π-calculus [15]). The main focus is not on communication, however, but on the ability of active processes to move between sites representing administrative domains; the calculus thereby extends the notion of mobility found in Java [13] where only passive code can be moved between sites. Both processes and sites are modelled as ambients; their ability to move around is governed by the capabilities possessed. The calculus was introduced in [7] and has been studied extensively [6, 8, 9, 14, 18, 19, 22]. We refer to Section 2 for a review of the ambient calculus.

Since processes may evolve when moving around, the structure of a system of ambients is very dynamic. In Section 3 we therefore develop a control flow analysis [17] for predicting the set of processes that may turn up inside a given ambient. This takes the form of defining a flow logic [16] for checking whether or not a control flow estimate (as might have been produced by a control flow analysis) is indeed acceptable; in the absence of higher-order features in the ambient calculus, this amounts to a syntax-directed definition of a number of judgements. The analysis combines the ability to handle communication (in the manner of analyses for the π-calculus [2, 3]) with the ability to handle movement (in the manner of an analysis for the communication-free fragment [18]). Semantic correctness is established by proving that all acceptable analyses are semantically sound (by means of a subject-reduction result in the manner of type systems).

On the algorithmic side we show that there always is a least control flow estimate and that it can be computed in polynomial time; this takes the form of generating a set of constraints that is then solved by a worklist algorithm [17].

In [7] the communication-free fragment of the ambient calculus is used to model and study a firewall where only agents knowing the required passwords are supposed to enter; indeed, assuming fairness, it is shown that all agents making correct use of all the passwords will in fact enter. However, in the interest of security and safety of systems, it is at least as important to ensure that an attacker not knowing any of the passwords cannot possibly enter; we shall say that the firewall is protective when this is the case. As an example, a protective firewall cannot contain trap doors or other ways of circumventing the protection offered by the passwords.

The difficulty of course is, that there are an infinity of attackers that do not know the passwords, and that it seems infeasible (and indeed undecidable) to perform automatic tests that will guard against all of these. To overcome these problems we change in Section 4 the "level of granularity" of our observations to coincide with those of the control flow analysis. We then prove that there is a process, called a hardest attacker, such that:

> If there exists a control flow estimate that shows the inability of the hardest attacker to enter, then none of the infinitely many attackers can enter unless they contain at least one of the passwords.

The ability to identify hardest attackers is perhaps comparable to the ability to identify hardest problems in given complexity classes. To argue the case in less technical jargon, consider the following "folk theorem":

> Testing[1] can prove the presence of bugs but never their absence.

Unfortunately, this has lead to the wide-spread belief that no experimentation with software can be used for formally validating software. The technical results presented here, generalising those of [18], provide a concrete instance of the rather different, and more useful, insight:

> Testing[2] can prove the absence of bugs but never their presence.

Expanding the area of applicability of this insight will likely lead to fundamental changes in the validation of software used in security oriented applications. The ability to extend the analysis to the existing software base, perhaps involving legacy code of "unknown" origin, offers a level of guarantee well above that of other formal approaches.

2 Mobile Ambients

Syntax. The presentation of the ambient calculus as given in [7] actually defines a "pre-syntax" for ambients. One aspect of this is that not all the defined ambient expressions are meaningful and hence a type system [8] is needed to rule out the undesired elements; the other aspect is that some of the clarifications made in the type system could in fact equally well be performed in the abstract syntax. To

[1] Testing in the sense of *dynamically running* a program on a number of inputs.
[2] Testing in the sense of *statically analysing* a program on a number of inputs.

$P ::= (\nu\, n^{\mu})P$	restriction	$M ::= \mathsf{in}^{l^{i}} N$		enter N
$\mid \mathbf{0}$	inactivity	$\mid \mathsf{out}^{l^{o}} N$		exit N
$\mid P \mid P'$	composition	$\mid \mathsf{open}^{l^{p}} N$		open N
$\mid !P$	replication	$\mid x$		variable
$\mid N^{l^{a}}[P]$	ambient	$\mid \epsilon$		null
$\mid M.P$	movement	$\mid M.M'$		path
$\mid \langle M \rangle^{l^{c}}$	output of capability			
$\mid \langle\!\langle N \rangle\!\rangle^{l^{n}}$	output of name	$N ::= n$		name
$\mid (x^{\beta^{c}}).P$	input of capability	$\mid u$		variable
$\mid (\!(u^{\beta^{n}})\!).P$	input of name			

Table 1. Abstract syntax.

avoid the artificial problem of devising semantics and static analysis for blatantly meaningless expressions we shall use a slightly more refined syntax that makes some of the distinctions of the type system.

The syntax of ambients in Table 1 is built around three syntactic categories: a class of processes, ranged over by $P \in \mathbf{Proc}$, a class of capabilities, ranged over by $M \in \mathbf{Cap}$, and a class of namings, ranged over by $N \in \mathbf{Nam}$. We follow [7] in distinguishing between names (introduced by the restriction operator known from process algebras) and variables (introduced by input statements); we also distinguish between variables used for holding capabilities, ranged over by $x \in \mathbf{Var}^{c}$, and variables used for holding names, ranged over by $u \in \mathbf{Var}^{n}$. We explain the constructs below.

First we consider processes (ignoring the superscript annotations in Table 1). Borrowing from the π-calculus [15] local scope is managed using the restriction operator. Also there is the inactive process, the parallel composition of two processes and a replicated process that is allowed to unfold to arbitrarily many ("infintely many") copies of the process. The next two constructs are unique to the ambient calculus. An ambient is a process operating inside a named border. Movement of ambients is governed by capabilities (explained below) and includes the ability for an ambient to move out of an enclosing ambient and for an ambient to move into a sibling ambient. Output of capabilities and names is much as in the π-calculus except that the channel is implicit and embedded in the enclosing ambient itself. Similarly for input of capabilities and names. As usual, trailing inactive actions will often be omitted from examples.

Next we consider capabilities and namings (once more ignoring the superscript annotations in Table 1). The in-capability directs the enclosing ambient to enter a sibling named N; this is illustrated in Figure 1 and will be explained in detail when we consider the semantics below. The out-capability directs the enclosing ambient to move out of its parent named N. The open-capability dissolves the border around a sibling ambient named N. Since capabilities can be communicated we also need variables ranging over them. Capabilities include the null capability as well as the sequential composition of capabilities describing a path to the desired destination. Namings are names but since they can be communicated we also need variables ranging over them.

Annotations. Let us now return to the two kinds of superscript annotations used in the syntax of Table 1. One class of annotations is composed of the *stable names*, ranged over by $\mu \in \mathbf{SNam}$, for names occurring in restrictions, and the *binders*, ranged over by $\beta^{c} \in \mathbf{Bnd}^{c}$ and $\beta^{n} \in \mathbf{Bnd}^{n}$, for variables occurring in input actions; we occasionally use $\beta \in \mathbf{Bnd} = \mathbf{Bnd}^{c} \cup \mathbf{Bnd}^{n}$. These annotations are necessary

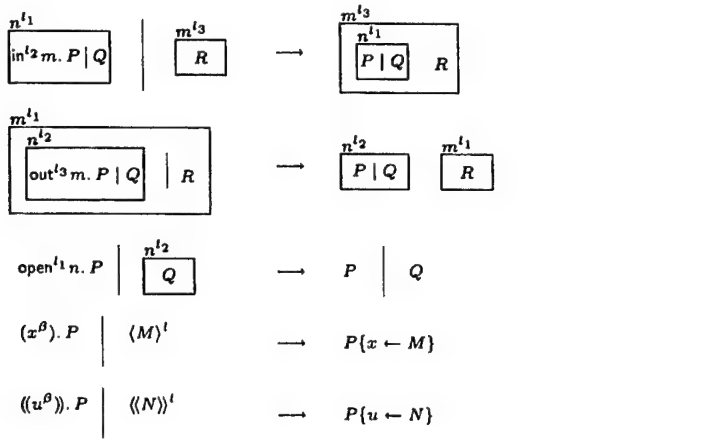


Fig. 1. Pictorial representation of the basic reduction rules.

for our analysis because the semantics of the ambient calculus borrows from the π-calculus in allowing α-conversion of bound names and variables. As an example, suppose we consider the ambient system

$$(\nu\, n)(\nu\, m)n[m[\mathbf{0}]]$$

and that our control flow estimate correctly says that m occurs inside n but not vice versa; unfortunately, this makes little sense since α-conversion allows us to change the above ambient system to

$$(\nu\, m)(\nu\, n)m[n[\mathbf{0}]]$$

and now the control flow estimate is no longer correct. To circumvent this problem we ensure that control flow estimates always refer to stable names and binders. Continuing the example, when we consider

$$(\nu\, n^{\mathsf{N}})(\nu\, m^{\mathsf{M}})n[m[\mathbf{0}]]$$

we say that M occurs inside N; this then remains correct for the α-converted system

$$(\nu\, m^{\mathsf{N}})(\nu\, n^{\mathsf{M}})m[n[\mathbf{0}]]$$

since stable names are never changed by α-conversion. Indeed, one way to understand the distinction between names and stable names is to regard the stable names as static representations of the names arising dynamically. The considerations for variables and binders are analogous.

The other class of annotations used in Table 1 are the labels. They assist in developing the control flow analysis by being able to precisely pin-point program points inside the ambient system; for this purpose it would be natural to let all labels be distinct since indeed all program points are distinct. As an example, in a system like

$$n[m^1[\mathsf{in}\, m] \mid m^2[\mathsf{out}\, n]]$$

this will allow us to distinguish the two occurrences of an ambient named m. As we shall see, labels allow us to control the complexity (and precision) of the analysis by treating one or more program points alike; for this purpose it may be appropriate

only to use a few labels. Indeed, one way to understand the use of labels is to regard labels as amalgamations of a number of program points.

We use $l \in \mathbf{Lab}$ to range over the set of labels. More specifically, we view labels as being defined by

$$\mathbf{Lab} = \mathbf{Lab}^a \cup \mathbf{Lab}^i \cup \mathbf{Lab}^o \cup \mathbf{Lab}^p \cup \mathbf{Lab}^c \cup \mathbf{Lab}^n$$

and use $l^a \in \mathbf{Lab}^a$ to annotate ambients, $l^i \in \mathbf{Lab}^i$ to annotate in-capabilities, $l^o \in \mathbf{Lab}^o$ to annotate out-capabilities, $l^p \in \mathbf{Lab}^p$ to annotate open-capabilities, $l^c \in \mathbf{Lab}^c$ to annotate the output of capabilities, and $l^n \in \mathbf{Lab}^n$ to annotate the output of names.

We shall assume that the sets of labels, $\mathbf{Lab}$, stable names, $\mathbf{SNam}$, binders for the input of capabilities, $\mathbf{Bnd}^c$, and binders for the input of names, $\mathbf{Bnd}^n$, are pairwise disjoint. It may aid the intution to assume that the different sets of labels, listed above, are also pairwise disjoint. Finally, we assume that all the sets mentioned are non-empty; for any given program they can always be assumed to be finite since the semantics below does not create new labels, stable names or binders.

We write $\underline{\mathrm{fn}}(P)$ for the set of *free names* of P and similarly for M and N. Similarly we write $\underline{\mathrm{fv}}(P)$ for the set of *free variables* of P (and similarly for M and N); a process is *closed* whenever it has no free variables, i.e. $\underline{\mathrm{fv}}(P) = \emptyset$, but may of course contain free names. Let $n_\star$ be a distinguished name and $l_\star$ a distinguished label; then the *programs* of interest are ambients of the form $n_\star^{l_\star}[P_\star]$ where $P_\star$ is closed, where $n_\star \notin \underline{\mathrm{fn}}(P_\star)$ and where $l_\star$ does not occur in $P_\star$.

Example 1. Consider the following example from [7] for illustrating how an agent crosses a firewall using the prearranged passwords (or secret keys) $\mathtt{k}$, $\mathtt{k}'$ and $\mathtt{k}''$:

$$\begin{array}{ll} \mathit{Firewall}: & (\nu\, \mathtt{w}^{w})\mathtt{w}^{\mathrm{A}}[\mathtt{k}^{\mathrm{B}}[\mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k}'.\,\mathsf{in}^3\mathtt{w}] \mid \mathsf{open}^4\mathtt{k}'.\,\mathsf{open}^5\mathtt{k}''.P] \\ \mathit{Agent}: & \mathtt{k}'^{\mathrm{C}}[\mathsf{open}^6\mathtt{k}.\,\mathtt{k}''^{\mathrm{D}}[Q]] \end{array}$$

The program of interest is $n_\star^{l_\star}[\mathit{Firewall} \mid \mathit{Agent}]$. We use typewriter font for names, italics for stable names, roman for ambient labels, and numbers for labels of capabilities. □

Semantics. The semantics is given by a structural congruence relation $P \equiv Q$ and a reduction relation $P \to Q$ in the manner of the π-calculus. The congruence relation is inductively defined by the axioms and rules of Table 2; apart from a few differences noted below it is a straightforward modification of a table in [7]. The axioms and rules in the left hand column ensure that the relation is an equivalence relation, that it is a congruence, that parallel composition is commutative and associative with the inactive process as a unit; they also describe the behaviour of replication.

The rules and axioms in the right hand column of Table 2 allow us to change the placement of restriction operators. In the axiom for $(\nu\, n^{\mu_n})(\nu\, m^{\mu_m})P$ we have added the side condition "$\underline{\text{if}}\ n \neq m$" to ensure that the association between names and stable names is not modified by the structural congruence. Next we have axioms for α-conversion; here we write $P\{n \leftarrow m\}$ for the process that is as P but with all free occurrences of n replaced by m (taking care to α-convert so as to avoid the capture of m by any restriction operator). The final two axioms control the expansion of capabilities.

The reduction relation is inductively defined by the axioms and rules of Table 3. It is as in [7] and a pictorial representation of the five basic axioms is given in Figure

$P \equiv P$	$(\nu n^{\mu_n})(\nu m^{\mu_m})P \equiv (\nu m^{\mu_m})(\nu n^{\mu_n})P$
$P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R$	$\underline{\text{if}}\ n \neq m$
$P \equiv Q \Rightarrow Q \equiv P$	
	$(\nu n^{\mu})(P \mid Q) \equiv P \mid (\nu n^{\mu})Q$
$P \equiv Q \Rightarrow (\nu n^{\mu})P \equiv (\nu n^{\mu})Q$	$\underline{\text{if}}\ n \notin \underline{\text{fn}}(P)$
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	$(\nu n^{\mu})(m^{l}[P]) \equiv m^{l}[(\nu n^{\mu})P]$
$P \equiv Q \Rightarrow !P \equiv !Q$	$\underline{\text{if}}\ n \neq m$
$P \equiv Q \Rightarrow N^{l}[P] \equiv N^{l}[Q]$	
$P \equiv Q \Rightarrow M.P \equiv M.Q$	$(\nu n^{\mu})P \equiv (\nu m^{\mu})(P\{n \leftarrow m\})$
$P \equiv Q \Rightarrow (x^{\beta^{c}}).P \equiv (x^{\beta^{c}}).Q$	$\underline{\text{if}}\ m \notin \underline{\text{fn}}(P)$ (α-renaming)
$P \equiv Q \Rightarrow ((u^{\beta^{n}})).P \equiv ((u^{\beta^{n}})).Q$	$(x^{\beta^{c}}).P \equiv (x'^{\beta^{c}}).(P\{x \leftarrow x'\})$
	$\underline{\text{if}}\ x' \notin \underline{\text{fv}}(P)$ (α-renaming)
$P \mid Q \equiv Q \mid P$	$((u^{\beta^{n}})).P \equiv ((u'^{\beta^{n}})).(P\{u \leftarrow u'\})$
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	$\underline{\text{if}}\ u' \notin \underline{\text{fv}}(P)$ (α-renaming)
$P \mid \mathbf{0} \equiv P$	
	$(M.M').P \equiv M.M'.P$
$!P \equiv P \mid !P$	$\epsilon.P \equiv P$
$!\mathbf{0} \equiv \mathbf{0}$	
$(\nu n^{\mu})\mathbf{0} \equiv \mathbf{0}$	

Table 2. Structural congruence.

1. The remaining rules ensure that reduction can take place in the contexts of restrictions, ambients and parallel compositions and that the structural congruence can freely be used to rearrange ambient expressions. Note that no internal reduction can take place "inside" movement or input prefixes. Also note that in each reduction, exactly one of the basic axioms is used.

Example 2. We have the following sequence of reduction steps for $n_{\star}^{l_{\star}}[\mathit{Firewall} \mid \mathit{Agent}]$; in each step we have underlined the capability to be executed next and we have assumed that $\mathtt{w} \notin \underline{\text{fn}}(Q)$.

$$
\begin{aligned}
& n_{\star}^{l_{\star}}[(\nu\,\mathtt{w}^{w})\mathtt{w}^{\mathrm{A}}[\mathtt{k}^{\mathrm{B}}[\underline{\mathsf{out}^{1}\mathtt{w}}.\,\mathsf{in}^{2}\mathtt{k}'.\,\mathsf{in}^{3}\mathtt{w}] \mid \mathsf{open}^{4}\mathtt{k}'.\,\mathsf{open}^{5}\mathtt{k}''.\,P] \mid \mathtt{k}'^{\mathrm{C}}[\mathsf{open}^{6}\mathtt{k}.\,\mathtt{k}''^{\mathrm{D}}[Q]]] \\
\rightarrow\ & n_{\star}^{l_{\star}}[(\nu\,\mathtt{w}^{w})(\mathtt{k}^{\mathrm{B}}[\underline{\mathsf{in}^{2}\mathtt{k}'}.\,\mathsf{in}^{3}\mathtt{w}] \mid \mathtt{w}^{\mathrm{A}}[\mathsf{open}^{4}\mathtt{k}'.\,\mathsf{open}^{5}\mathtt{k}''.\,P] \mid \mathtt{k}'^{\mathrm{C}}[\mathsf{open}^{6}\mathtt{k}.\,\mathtt{k}''^{\mathrm{D}}[Q]])] \\
\rightarrow\ & n_{\star}^{l_{\star}}[(\nu\,\mathtt{w}^{w})(\mathtt{w}^{\mathrm{A}}[\mathsf{open}^{4}\mathtt{k}'.\,\mathsf{open}^{5}\mathtt{k}''.\,P] \mid \mathtt{k}'^{\mathrm{C}}[\mathtt{k}^{\mathrm{B}}[\mathsf{in}^{3}\mathtt{w}] \mid \underline{\mathsf{open}^{6}\mathtt{k}}.\,\mathtt{k}''^{\mathrm{D}}[Q]])] \\
\rightarrow\ & n_{\star}^{l_{\star}}[(\nu\,\mathtt{w}^{w})(\mathtt{w}^{\mathrm{A}}[\mathsf{open}^{4}\mathtt{k}'.\,\mathsf{open}^{5}\mathtt{k}''.\,P] \mid \mathtt{k}'^{\mathrm{C}}[\underline{\mathsf{in}^{3}\mathtt{w}} \mid \mathtt{k}''^{\mathrm{D}}[Q]])] \\
\rightarrow\ & n_{\star}^{l_{\star}}[(\nu\,\mathtt{w}^{w})\mathtt{w}^{\mathrm{A}}[\underline{\mathsf{open}^{4}\mathtt{k}'}.\,\mathsf{open}^{5}\mathtt{k}''.\,P \mid \mathtt{k}'^{\mathrm{C}}[\mathtt{k}''^{\mathrm{D}}[Q]]]] \\
\rightarrow\ & n_{\star}^{l_{\star}}[(\nu\,\mathtt{w}^{w})\mathtt{w}^{\mathrm{A}}[\underline{\mathsf{open}^{5}\mathtt{k}''}.\,P \mid \mathtt{k}''^{\mathrm{D}}[Q]]] \\
\rightarrow\ & n_{\star}^{l_{\star}}[(\nu\,\mathtt{w}^{w})\mathtt{w}^{\mathrm{A}}[P \mid Q]]
\end{aligned}
$$

The transition sequence shows that the firewall (which has the private name $\mathtt{w}$) sends out the pilot ambient named $\mathtt{k}$; since the agent knows the right passwords, and is in the right form, the pilot ambient can enter the agent and then guide it inside the firewall. □

Properties of the semantics. Recall that the programs of interest are ambients of the form $n_{\star}^{l_{\star}}[P_{\star}]$ where $P_{\star}$ is closed (and hence contains no free variables but may contain free names), $n_{\star} \notin \underline{\text{fn}}(P_{\star})$ and $P_{\star}$ does not contain $l_{\star}$. It follows that only closed expressions are reduced and that no new names become free and that no new labels come into existence. Because of the structural congruence it is not the case that programs evolve into programs (in particular processes of the form $n_{\star}^{l_{\star}}[\cdots]$); to achieve this we could restrict the use of the congruence in the semantics. Instead, we shall use the following result showing that programs evolve into processes that are congruent to programs.

$P \to Q \Rightarrow (\nu\, n^{\mu})P \to (\nu\, n^{\mu})Q$	$n^{l_1}[\mathsf{in}^{l_2} m.\, P \mid Q] \mid m^{l_3}[R] \to m^{l_3}[n^{l_1}[P \mid Q] \mid R]$
$P \to Q \Rightarrow n^{l}[P] \to n^{l}[Q]$	$m^{l_1}[n^{l_2}[\mathsf{out}^{l_3} m.\, P \mid Q] \mid R] \to n^{l_2}[P \mid Q] \mid m^{l_1}[R]$
$P \to Q \Rightarrow P \mid R \to Q \mid R$	$\mathsf{open}^{l_1} n.\, P \mid n^{l_2}[Q] \to P \mid Q$
$\left.\begin{array}{l} P \equiv P' \\ P' \to Q' \\ Q' \equiv Q \end{array}\right\} \Rightarrow P \to Q$	$(x^{\beta}).\, P \mid \langle M \rangle^{l} \to P\{x \leftarrow M\}$
	$(\!(u^{\beta})\!).\, P \mid (\!(N)\!)^{l} \to P\{u \leftarrow N\}$

Table 3. Reduction relation.

Fact 1. Programs evolve to programs (modulo the congruence):

If P is congruent to a program and $P \to^{*} Q$ then also Q is congruent to a program.

Proof. We first investigate what it means for a process to be congruent to a program. For this we extend the syntax of Table 1 with

$$
\begin{array}{lll}
T ::= (\nu\, n^{\mu})T & S ::= (\nu\, n^{\mu})S & \text{if } n \neq n_{\star} \\
\quad \mid \mathbf{0} & \quad \mid S \mid T & \\
\quad \mid T \mid T' & \quad \mid T \mid S & \\
\quad \mid\, !T & \quad \mid n_{\star}^{l_{\star}}[P] & \text{if } n_{\star}^{l_{\star}}[P] \text{ is a program} \\
\quad \mid L.T & \quad \mid L.S &
\end{array}
$$

$$L ::= \epsilon \mid L.L'$$

Let **Triv** be the set of processes described by T and let **Ser** be the set of processes described by S; the former clearly contains the inactive process $\mathbf{0}$ and the latter clearly contains all programs. We now show that **Triv**, respectively **Ser**, is the closure of the set $\{\mathbf{0}\}$, respectively the set of programs, under the congruence.

It is immediate to prove by induction in L that $L.T \equiv T$ and $L.S \equiv S$. It is also immediate to prove by induction in T that $T \equiv \mathbf{0}$; hence all processes in **Triv** are congruent to $\mathbf{0}$. Furthermore, by induction in S one can show that $S \equiv n_{\star}^{l_{\star}}[P]$ for some P such that $n_{\star}^{l_{\star}}[P]$ is a program; hence all processes in **Ser** are congruent to programs.

For the opposite inclusions we prove that if $P \equiv Q$ and $P \in$ **Triv** then $Q \in$ **Triv**, and similarly that if $P \equiv Q$ and $Q \in$ **Triv** then $P \in$ **Triv**, by induction in the inference; it follows that **Triv** is the set of processes that are congruent to $\mathbf{0}$. Next we prove that if $P \equiv Q$ and $P \in$ **Ser** then $Q \in$ **Ser**, and similarly that if $P \equiv Q$ and $Q \in$ **Ser** then $P \in$ **Ser**, by induction in the inference; hence **Ser** is the set of processes that are congruent to programs.

We now turn to the statement of the fact: if $P \in$ **Ser** and $P \to^{*} Q$ then $Q \in$ **Ser**. We proceed by induction in the length of the derivation. In the induction step, $P \to^{*} R \to Q$ with $R \in$ **Ser**, we consider the place where the basic axiom is used for establishing $R \to Q$. Since a process of the form T cannot contain any labels or binders the basic axiom used for $R \to Q$ cannot involve any subprocess of the form T. It follows that the basic axiom must take place inside the (necessarily unique) occurrence of $n_{\star}^{l_{\star}}[\cdots]$ in R. Inspection of the five basic axioms then immediately establish the result. □

It should be clear that the syntactic annotations in no way influence the semantics. We can make this precise as follows. Let $\mu_{\bullet}$ be a distinguished stable name, let $\beta_{\bullet}^{c}$

and $\beta^n_\bullet$ be distinguished binders, and let $l^a_\bullet$, $l^i_\bullet$, $l^o_\bullet$, $l^p_\bullet$, $l^c_\bullet$ and $l^n_\bullet$ be distinguished labels. Given a process P write $\lfloor P \rfloor$ for the process where all stable names, binders and labels are replaced by the appropriate distinguished stable names, binders and labels.

Fact 2. The semantics is invariant under annotations:

> If $P \rightarrow^* Q$ and $\lfloor P \rfloor = \lfloor P' \rfloor$ then there exists Q' such that $P' \rightarrow^* Q'$ and $\lfloor Q \rfloor = \lfloor Q' \rfloor$.

Proof. We first consider the similar statement for the congruence:

> If $P \equiv Q$ and $\lfloor P \rfloor = \lfloor P' \rfloor$ then there exists Q' such that $P' \equiv Q'$ and $\lfloor Q \rfloor = \lfloor Q' \rfloor$; similarly if $P \equiv Q$ and $\lfloor Q \rfloor = \lfloor Q' \rfloor$ then there exists P' such that $P' \equiv Q'$ and $\lfloor P \rfloor = \lfloor P' \rfloor$.

It is proved by induction in the inference tree for $P \equiv Q$.

We then prove the statement of the fact by induction in the length of the derivation $P \rightarrow^* Q$. For the induction step $P \rightarrow^* R \rightarrow Q$ we proceed by induction in the shape of the inference tree for $R \rightarrow Q$. □

3 Control Flow Analysis

Immediate constituents of ambients. The main aim of the control flow analysis is to obtain the following information for each ambient: *(i)* which ambients *may* be immediately contained in it, *(ii)* which capabilities *may* it perform, *(iii)* which input actions *may* be performed immediately inside it, and *(iv)* which output actions *may* be performed immediately inside it. An ambient will be identified by its label $l^a \in \mathbf{Lab}^a$, an in-capability by its label $l^i \in \mathbf{Lab}^i$, an out-capability by its label $l^o \in \mathbf{Lab}^o$, an open-capability by its label $l^p \in \mathbf{Lab}^p$, an input of a capability by its binder[3] $\beta^c \in \mathbf{Bnd}^c$, an input of a name by its binder $\beta^n \in \mathbf{Bnd}^n$, the output of a capability by its label $l^c \in \mathbf{Lab}^c$, and the output of a name by its label $l^n \in \mathbf{Lab}^n$.

The analysis records this information in the following component:

$$I \in \mathbf{InAmb} = \mathbf{Lab}^a \rightarrow \mathcal{P}\left(\begin{array}{l}\mathbf{Lab}^a \cup \mathbf{Lab}^i \cup \mathbf{Lab}^o \cup \mathbf{Lab}^p \cup \\ \mathbf{Lab}^c \cup \mathbf{Lab}^n \cup \mathbf{Bnd}^c \cup \mathbf{Bnd}^n\end{array}\right)$$

When specifying the analysis we shall also use the "inverse" mapping

$$I^{-1} : (\mathbf{Lab} \cup \mathbf{Bnd}) \rightarrow \mathcal{P}(\mathbf{Lab}^a)$$

that returns the set of ambients in which the given ambient, capability, input or output might occur; formally $z \in I(l^a)$ if and only if $l^a \in I^{-1}(z)$. Later we shall write $z \in I^+(l)$ to mean that there exists $l_1, \cdots, l_n$ (for $n \geq 1$) such that $l = l_1$, $z = l_n$, and $\forall i < n : l_{i+1} \in I(l_i)$.

[3] This actually confuses binders with labels; it would be notationally purer, but somewhat heavier, to demand input actions to be annotated not only with a binder but also with a label.

Stable names of ambients and capabilities. Ambients and capabilities have stable names associated with them and to keep track of this information the analysis also contains the following component:

$$H \in \mathbf{HNam} = (\mathbf{Lab}^a \cup \mathbf{Lab}^i \cup \mathbf{Lab}^o \cup \mathbf{Lab}^p) \to \mathcal{P}(\mathbf{SNam})$$

As above we shall also use the "inverse mapping"

$$H^{-1} : \mathbf{SNam} \to \mathcal{P}(\mathbf{Lab}^a \cup \mathbf{Lab}^i \cup \mathbf{Lab}^o \cup \mathbf{Lab}^p)$$

that returns the set of ambients that might have the given stable name; formally $\mu \in H(l)$ if and only if $l \in H^{-1}(\mu)$. The information in H is needed to determine the ambients operated upon by the capabilities so as to accurately update the contents of I.

Naming environment. The association between free names and variables, and their stable names and binders, is expressed by a naming environment:

$$\begin{aligned} me \in \mathbf{MEnv} = (\mathbf{Nam} \cup \mathbf{Var}^c \cup \mathbf{Var}^n) &\to_{\mathsf{fin}} (\mathbf{SNam} \cup \mathbf{Bnd}^c \cup \mathbf{Bnd}^n) \\ &\underline{\text{such that}}\ me(n) \in \mathbf{SNam}, me(x) \in \mathbf{Bnd}^c, me(u) \in \mathbf{Bnd}^n \end{aligned}$$

Here we impose the condition that the marker environment "preserves the types" of names, variables ranging over capabilities and variables ranging over names.

We shall write $me_\star$ for the initial naming environment for the program $n_\star^{l_\star}[P_\star]$ of interest and $dom(me_\star)$ for its finite domain. Recall that for $n_\star^{l_\star}[P_\star]$ to be a program we demand that $P_\star$ is closed, that $n_\star \notin \underline{\mathrm{fn}}(P_\star)$ and that $P_\star$ does not contain $l_\star$. For $(me_\star, n_\star^{l_\star}[P_\star])$ to constitute a program of interest we then demand that:

- $n_\star^{l_\star}[P_\star]$ is a program,
- $me_\star$ defines all the free names of $n_\star^{l_\star}[P_\star]$, i.e. $\underline{\mathrm{fn}}(n_\star^{l_\star}[P_\star]) \subseteq dom(me_\star)$, and $me_\star$ does not define any variables, i.e. $dom(me_\star) \subseteq \mathbf{Nam}$, and
- the stable name $\mu_\star$ is distinguished, does not occur in $P_\star$, and is only possessed by $n_\star$, i.e. $me_\star^{-1}(\mu_\star) = \{n_\star\}$.

(Here we write $me_\star^{-1}(\mu)$ for the set $\{n \mid me_\star(n) = \mu\}$ of names mapped to μ.)

Communication and stable capabilities. The *environment* R is responsible for collecting the values that can be bound to the variables as result of an *input action.* The variables are represented by their binders. In the case of input of names it is natural to represent the name by its stable name. In a similar way, in the case of input of capabilities, we shall represent the capability by its stable capability:

$$\begin{aligned} \tilde{m} &\in \mathbf{SCap} \\ \tilde{m} &::= \mathsf{in}^{l^i} \mid \mathsf{out}^{l^o} \mid \mathsf{open}^{l^p} \end{aligned}$$

There are no stable capabilities corresponding to null capabilities (ϵ) and paths ($M_1.M_2$); instead the analysis will break[4] them into the set of constituent in-, out-, and open-capabilities.

The *communication box* C is responsible for collecting the effects of the *output actions* taking place immediately inside the ambient: again the ambient is identified

[4] Clearly a more precise analysis can be devised; however, we choose the simpler approach so that the constraint solver of Subsection 3.3 only needs to operate in a finite universe and hence can compute solutions explicitly in polynomial time.

by its label and the value being communicated will be a stable capability or a stable name:

$$R = (R^c, R^n) \in \mathbf{Env} = (\mathbf{Bnd}^c \to \mathcal{P}(\mathbf{SCap})) \times (\mathbf{Bnd}^n \to \mathcal{P}(\mathbf{SNam}))$$
$$C = (C^c, C^n) \in \mathbf{Comm} = (\mathbf{Lab}^a \to \mathcal{P}(\mathbf{SCap})) \times (\mathbf{Lab}^a \to \mathcal{P}(\mathbf{SNam}))$$

Also the information in R and C will be needed to accurately update the contents of I in the presence of communication.

Example 3. Consider the program $n_\star^{l_\star}[\mathit{Firewall} \mid \mathit{Agent}]$ of Example 1 and the following analysis estimate (where the initial naming environment $me_\star$ maps the names $n_\star$, k, k′ and k″ to $\mu_\star$, k, k' and k'', respectively):

$$I(l_\star) = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$$
$$I(\mathrm{A}) = \{1, 2, 3, 4, 5, 6, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$$
$$I(\mathrm{B}) = \{1, 2, 3\}$$
$$I(\mathrm{C}) = \{1, 2, 3, 6, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$$
$$I(\mathrm{D}) = \emptyset$$

label	$l_\star$	A	B	C	D	1	2	3	4	5	6
H	$\{\mu_\star\}$	$\{w\}$	$\{k\}$	$\{k'\}$	$\{k''\}$	$\{w\}$	$\{k'\}$	$\{w\}$	$\{k'\}$	$\{k''\}$	$\{k\}$

This shows that the ambient labelled A might perform transitions consuming any of the capabilities labelled 1–6 and that it might contain any of the ambients labelled A–D; in particular it might contain the ambient labelled C indicating that the agent might enter the firewall – and as shown in Example 2 this indeed happens.

No communication is taking place and it is therefore safe to set $R^c(\beta^c) = \emptyset$ and $R^n(\beta^n) = \emptyset$ for all binders and to set $C^c(l^a) = \emptyset$ and $C^n(l^a) = \emptyset$ for all labels. □

3.1 The acceptability relation

The acceptability of a control flow estimate is defined by the following four predicates (defined in Tables 4 and 5 and explained below):

$(I, H, C, R) \models^l_{me} P$	for checking a process $P \in \mathbf{Proc}$;
$(I, H, C, R) \rhd_{me} M : \tilde{M}$	for translating a capability $M \in \mathbf{Cap}$ into a set $\tilde{M} \in \mathcal{P}(\mathbf{SCap})$ of stable capabilities;
$(I, H, C, R) \models\!\!\models_{me} N : \tilde{N}$	for decoding a naming $N \in \mathbf{Nam}$ into a set $\tilde{N} \in \mathcal{P}(\mathbf{SNam})$ of stable names;
$(I, H, C, R) \models^l \tilde{m}$	for checking a stable capability $\tilde{m} \in \mathbf{SCap}$.

Analysis of processes. Table 4 gives a simple syntax-directed definition of what it means for an analysis estimate (I, H, C, R) to be acceptable for the process P. The predicate is defined relative to the current naming environment me and the current label l of the enclosing ambient. The naming environment is updated whenever we pass through a restriction operator or an input and the label is updated whenever we pass inside a new ambient. Note that the analysis cannot distinguish between whether a process occurs only once or many times: $!P$ and P are analysed in the same way (as are $P \mid P$ and P).

The clause for ambients $N^{l^a}[P]$ first checks the subprocess P using the appropriate naming environment and label. It then demands that the label of the ambient is

$$(I,H,C,R) \models^l_{me} (\nu n^\mu)P \quad \underline{\text{iff}} \quad (I,H,C,R) \models^l_{me[n \mapsto \mu]} P$$

$$(I,H,C,R) \models^l_{me} \mathbf{0} \quad \underline{\text{iff}} \quad \text{true}$$

$$(I,H,C,R) \models^l_{me} P \mid P' \quad \underline{\text{iff}} \quad (I,H,C,R) \models^l_{me} P \ \wedge \ (I,H,C,R) \models^l_{me} P'$$

$$(I,H,C,R) \models^l_{me} !P \quad \underline{\text{iff}} \quad (I,H,C,R) \models^l_{me} P$$

$$(I,H,C,R) \models^l_{me} N^{l^a}[P] \quad \underline{\text{iff}} \quad (I,H,C,R) \models^{l^a}_{me} P \ \wedge \ l^a \in I(l) \ \wedge \ (I,H,C,R) \models_{me} N : \tilde{N} \ \wedge \ \tilde{N} \subseteq H(l^a)$$

$$(I,H,C,R) \models^l_{me} M.P \quad \underline{\text{iff}} \quad (I,H,C,R) \models^l_{me} P \ \wedge \ (I,H,C,R) \rhd_{me} M : \tilde{M} \ \wedge \ \forall \tilde{m} \in \tilde{M} : (I,H,C,R) \models^l \tilde{m}$$

$$(I,H,C,R) \models^l_{me} \langle M \rangle^{l^c} \quad \underline{\text{iff}} \quad l^c \in I(l) \ \wedge \ (I,H,C,R) \rhd_{me} M : \tilde{M} \ \wedge \ \forall l^a \in I^{-1}(l^c) : C^c(l^a) \supseteq \tilde{M}$$

$$(I,H,C,R) \models^l_{me} (\!|N|\!)^{l^n} \quad \underline{\text{iff}} \quad l^n \in I(l) \ \wedge \ (I,H,C,R) \models_{me} N : \tilde{N} \ \wedge \ \forall l^a \in I^{-1}(l^n) : C^n(l^a) \supseteq \tilde{N}$$

$$(I,H,C,R) \models^l_{me} (x^{\beta^c}).P \quad \underline{\text{iff}} \quad (I,H,C,R) \models^l_{me[x \mapsto \beta^c]} P \ \wedge \ \beta^c \in I(l) \ \wedge \ \forall l^a \in I^{-1}(\beta^c) : C^c(l^a) \subseteq R^c(\beta^c)$$

$$(I,H,C,R) \models^l_{me} (\!(u^{\beta^n})\!).P \quad \underline{\text{iff}} \quad (I,H,C,R) \models^l_{me[u \mapsto \beta^n]} P \ \wedge \ \beta^n \in I(l) \ \wedge \ \forall l^a \in I^{-1}(\beta^n) : C^n(l^a) \subseteq R^n(\beta^n)$$

Table 4. Control flow analysis (for processes).

recorded as being inside the current label. Finally, it demands that the stable name of the ambient is recorded as being a name of the ambient. Intuitively, $\tilde{N}$ is the singleton $\{me(n)\}$ when N is n; this is made precise by Table 5 explained below.

As in Prolog, all free identifiers (like $\tilde{N}$ and $\tilde{M}$) on the right-hand sides of clauses are implicitly assumed to be existentially quantified; this means that whenever a clause is applied we are free to supply suitable values for these identifiers.

The clause for movement $M.P$ first checks the subprocess P using the appropriate naming environment and label. It then translates the capability M into the set of stable capabilities $\tilde{M}$. Intuitively $\tilde{M}$ is the singleton $\{\mathsf{in}^{l^i}\}$ when M is $\mathsf{in}^{l^i} n$ and similarly for the other capabilities; this is made precise by Table 5 explained below (that also takes care of associating the stable name of n with the label l^i).

The two clauses for output actions first record that the action may take place inside the enclosing ambient. The next step is to translate the capability M (resp. the name N) into a set of stable capabilities $\tilde{M}$ (resp. a set of stable names $\tilde{N}$) thereby making it independent of the context. The communication box C is then updated to record that each ambient l^a (including l) that could contain the output action will in fact witness the output of $\tilde{M}$ (resp. $\tilde{N}$).

Finally, the two clauses for input action first update the environment me to record the binding of the variable before analysing the subprocess. It is then ensured that the component I contains the appropriate binder (β^c or β^n) representing the input action. To determine the possible value being communicated (and hence bound to the variable represented by the binder) we have to consult the communication box of the enclosing ambient. So for all ambients l^a (including l) that might contain the

$(I,H,C,R) \rhd_{me} \mathsf{in}^{l^i} N : \tilde{M}$	iff	$(I,H,C,R) \Vdash_{me} N : \tilde{N} \wedge \tilde{M} \supseteq \{\mathsf{in}^{l^i}\} \wedge H(l^i) \supseteq \tilde{N}$
$(I,H,C,R) \rhd_{me} \mathsf{out}^{l^o} N : \tilde{M}$	iff	$(I,H,C,R) \Vdash_{me} N : \tilde{N} \wedge \tilde{M} \supseteq \{\mathsf{out}^{l^o}\} \wedge H(l^o) \supseteq \tilde{N}$
$(I,H,C,R) \rhd_{me} \mathsf{open}^{l^p} N : \tilde{M}$	iff	$(I,H,C,R) \Vdash_{me} N : \tilde{N} \wedge \tilde{M} \supseteq \{\mathsf{open}^{l^p}\} \wedge H(l^p) \supseteq \tilde{N}$
$(I,H,C,R) \rhd_{me} x : \tilde{M}$	iff	$\tilde{M} \supseteq R^c(me(x))$
$(I,H,C,R) \rhd_{me} \epsilon : \tilde{M}$	iff	$\tilde{M} \supseteq \emptyset$
$(I,H,C,R) \rhd_{me} M_1.M_2 : \tilde{M}$	iff	$(I,H,C,R) \rhd_{me} M_1 : \tilde{M}_1 \wedge (I,H,C,R) \rhd_{me} M_2 : \tilde{M}_2 \wedge \tilde{M} \supseteq \tilde{M}_1 \cup \tilde{M}_2$
$(I,H,C,R) \Vdash_{me} n : \tilde{N}$	iff	$\tilde{N} \supseteq \{me(n)\}$
$(I,H,C,R) \Vdash_{me} u : \tilde{N}$	iff	$\tilde{N} \supseteq R^n(me(u))$
$(I,H,C,R) \models^l \mathsf{in}^{l^i}$	iff	$l^i \in I(l) \wedge \forall l^a \in I^{-1}(l^i) : \forall \mu \in H(l^i) : \forall l^{a'} \in I^{-1}(l^a) : \forall l^{a''} \in I(l^{a'}) \cap H^{-1}(\mu) \cap \mathbf{Lab}^a : l^a \in I(l^{a''})$
$(I,H,C,R) \models^l \mathsf{out}^{l^o}$	iff	$l^o \in I(l) \wedge \forall l^a \in I^{-1}(l^o) : \forall \mu \in H(l^o) : \forall l^{a'} \in I^{-1}(l^a) \cap H^{-1}(\mu) : \forall l^{a''} \in I^{-1}(l^{a'}) : l^a \in I(l^{a''})$
$(I,H,C,R) \models^l \mathsf{open}^{l^p}$	iff	$l^p \in I(l) \wedge \forall l^a \in I^{-1}(l^p) : \forall \mu \in H(l^p) : \forall l^{a'} \in I(l^a) \cap H^{-1}(\mu) \cap \mathbf{Lab}^a : \forall l' \in I(l^{a'}) : l' \in I(l^a)$

Table 5. Control flow analysis (for capabilities and namings).

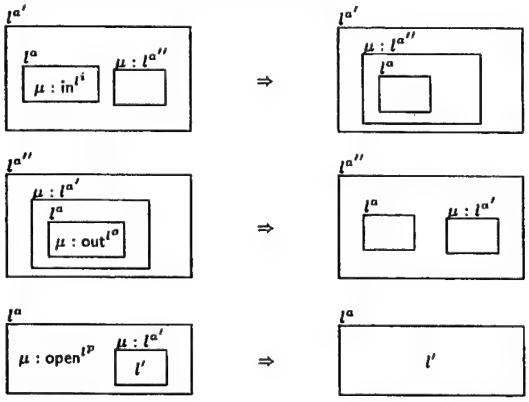


Fig. 2. Pictorial representation of the analysis of atomic capabilities.

input action we record that the contents of the communication box $C(l^a)$ can be a value of the binder and we use the environment R to capture this.

Translation of capabilities and namings. The first two parts of Table 5 translate capabilities and namings into a form where they are independent of the context. For capabilities we return (in $\tilde{M}$) the stable capability and in the H component the stable names relating to the capability. It would have been very natural to follow [18] in bypassing the H component for this purpose and to include the stable name in the stable capability; the reason for not doing so is to allow approximations that will yield faster analyses than possible using the approach of [18].

The entities recorded in R come into play in the clause for translating variables to stable capabilities and stable namings; as an example, for a capability variable x we consult *me* and R^c to determine the possible stable capabilities that x might stand for. The two clauses for null capabilities and paths show how capabilites are broken up into their atomic constituents; as mentioned earlier this is to facilitate the development of a simple constraint solver for implementing the analysis in polynomial time.

The form of the judgements for translating capabilities and namings combine the *verbose* and *succinct* forms of flow logic [16]. The verbose format, as used for the analysis of processes and capabilities, explicitly contains a record of the information as it pertains to all internal program points; this is part of the (I, H, C, R) component. The succinct format, to be specific the $\tilde{M}$ and $\tilde{N}$ components of the judgements for translation, directly expresses auxiliary information that is only of local interest. The use of succinct components frequently make specifications more readable and tend to give them the flavour of type systems. The relationship between verbose and succinct specifications is studied in [21].

Analysis of capabilities. The last part of Table 5 shows how to check stable capabilities against the control flow estimate (I, H, C, R). Figure 2 illustrates these clauses pictorially; the similarity between Figures 2 and 1 stresses the systematic way in which a control flow analysis may be developed from a formal semantics and we regard this as a strong point of our approach.

The clause for $\mathsf{in}^{l'}$ first ensures that the stable capability is properly recorded as part of the current ambient l. Then it ensures that all contexts l^a in which the capability could occur (and this clearly includes l) are properly recorded as being possible subambients of all sibling ambients $l^{a''}$ having a stable name μ associated with $\mathsf{in}^{l'}$. This involves quantifying over all possible parent ambients $l^{a'}$ and using the component H to obtain the stable name of the ambient that $l^{a''}$ indicates.

The clause for $\mathsf{out}^{l'}$ follows a similar pattern. First it ensures that the stable capability is recorded as part of the current ambient l. Next it ensures that all contexts l^a in which the capability could occur (and again this includes l) are properly recorded as being possible ambients in all the possible grandparents $l^{a''}$ provided that the parent $l^{a'}$ has a stable name μ associated with $\mathsf{out}^{l'}$.

For the stable capability $\mathsf{open}^{l'}$ we once again start by ensuring that it is properly recorded as part of the current ambient l. Then we consider all contexts l^a in which the capability could occur (and again this includes l) and find all subambients $l^{a'}$ having a stable name μ associated with $\mathsf{open}^{l'}$; these are opened by ensuring that whatever is included in the subambient $l^{a'}$ also occurs in the parent ambient l^a.

It is crucial to observe that we need to consult all possible contexts l^a in which the capability could occur and not just the obvious candidate l. This is because, in

order to establish semantic soundness, the analysis has to take into account that the current ambient might be dissolved by an open-capability.

Example 4. Let us check the condition

$$(I,H,C,R) \models^{\mathsf{A}}_{me} \mathtt{k}^{\mathsf{B}}[\mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k}'.\,\mathsf{in}^3\mathtt{w}]$$

that arises when checking that the analysis estimate (I,H,C,R) of Example 3 correctly validates the program $n^{l_\star}_\star[\mathit{Firewall} \mid \mathit{Agent}]$ of Example 1; here the naming environment me maps $n_\star$, $\mathtt{k}$, $\mathtt{k}'$, $\mathtt{k}''$ and $\mathtt{w}$ to $\mu_\star$, k, k', k'' and w, respectively. First we decide to let $\tilde{N}$ be $\{k\}$. We then need to check that $(I,H,C,R) \models_{me} \mathtt{k} : \{k\}$ (which follows from the choice of me), that $\{k\} \subseteq H(\mathrm{B})$ (which follows from Example 3), that $\mathrm{B} \in I(\mathrm{A})$ (which once more follows from Example 3) and that $(I,H,C,R) \models^{\mathsf{B}}_{me} \mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k}'.\,\mathsf{in}^3\mathtt{w}$ (see below).

To check that $(I,H,C,R) \models^{\mathsf{B}}_{me} \mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k}'.\,\mathsf{in}^3\mathtt{w}$ we first decide to let $\tilde{M}$ be $\{\mathsf{out}^1\}$. We then need to check that $(I,H,C,R) \rhd_{me} \mathsf{out}^1\mathtt{w} : \{\mathsf{out}^1\}$ (which follows from $(I,H,C,R) \models_{me} \mathtt{w} : \{w\}$ and $H(1) \supseteq \{w\}$), that $(I,H,C,R) \models^{\mathsf{B}} \mathsf{out}^1$ (see below) and that $(I,H,C,R) \models^{\mathsf{B}}_{me} \mathsf{in}^2\mathtt{k}'.\,\mathsf{in}^3\mathtt{w}$ (which amounts to twice repeating the checking procedure being illustrated for $\mathsf{out}^1\mathtt{w}$).

Finally, let us check that $(I,H,C,R) \models^{\mathsf{B}} \mathsf{out}^1$. First we check that $1 \in I(\mathrm{B})$ (using Example 3). For the second condition we have $l^a \in I^{-1}(1) = \{\mathrm{A},\mathrm{B},\mathrm{C}\}$ and $\mu \in H(1) = \{w\}$; for each of the choices for l^a we have $l^{a'} \in I^{-1}(l^a) \cap H^{-1}(w) = \{l_\star,\mathrm{A},\mathrm{C}\} \cap \{\mathrm{A},1,3\} = \{\mathrm{A}\}$ so the parent ambient $l^{a'}$ of l^a will always be A. The grandparent of l^a is $l^{a''} \in I^{-1}(\mathrm{A}) = \{l_\star,\mathrm{A},\mathrm{C}\}$ so the second condition amounts to checking that all of A, B and C are elements of all of $I(l_\star)$, $I(\mathrm{A})$ and $I(\mathrm{C})$ and clearly this is the case. □

3.2 Properties of the analysis

In the terminology of data flow analysis [17] the above analysis is *flow-insensitive* since we ignore the order in which the capabilities occur; also it is *context-insensitive* (or *monovariant*) since a capability is analysed in the same way for all contexts in which it occurs. We refer to [14, 19, 22] for more precise ways of analysing the communication-free fragment.

Semantic correctness. Having specified what it means for an analysis estimate (I,H,C,R) to be acceptable the next step is to show that the notion of acceptability is semantically meaningful. We begin by establishing some auxiliary properties.

Fact 3. The analysis enjoys the following monotonicity properties:

(i) If $(I,H,C,R) \models^{l_1}_{me} P$ and $I(l_1) \subseteq I(l_2)$ then $(I,H,C,R) \models^{l_2}_{me} P$.
(ii) If $(I,H,C,R) \rhd_{me} M : \tilde{M}_1$ and $\tilde{M}_1 \subseteq \tilde{M}_2$ then $(I,H,C,R) \rhd_{me} M : \tilde{M}_2$.
(iii) If $(I,H,C,R) \models_{me} N : \tilde{N}_1$ and $\tilde{N}_1 \subseteq \tilde{N}_2$ then $(I,H,C,R) \models_{me} N : \tilde{N}_2$.
(iv) If $(I,H,C,R) \models^{l_1} \tilde{m}$ and $I(l_1) \subseteq I(l_2)$ then $(I,H,C,R) \models^{l_2} \tilde{m}$.

Proof. It is immediate to prove *(iii)* by inspection of the two cases for N.

We then prove *(ii)* by inspection of the six cases for M and all but one case is immediate. In the case for $M_1.M_2$ we have $(I,H,C,R) \rhd_{me} M_1 : \tilde{M}'_1$ and $(I,H,C,R) \rhd_{me} M_2 : \tilde{M}'_2$ for some $\tilde{M}'_1$ and $\tilde{M}'_2$ such that $\tilde{M}_1 \supseteq \tilde{M}'_1 \cup \tilde{M}'_2$; it is then immediate that $\tilde{M}_2 \supseteq \tilde{M}'_1 \cup \tilde{M}'_2$ as was to be shown.

Next we prove *(iv)* by inspection of the three cases for $\tilde{m}$. All cases are immediate since the label l_i (for $i = 1, 2$) is only used to establish a fact of the form $l' \in I(l_i)$.

Finally we prove *(i)* by structural induction in P. This is straightforward because the label l_i (for $i = 1, 2$) is only used in recursive calls, to establish a fact of the form $l' \in I(l_i)$ or $\beta' \in I(l_i)$, or to invoke *(iv)*. □

To express the next fact we shall write $me_1 =_P me_2$ to mean that me_1 and me_2 are equal on the free names and variables of P; in a similar way we write $me_1 =_M me_2$ and $me_1 =_N me_2$ for capabilities M and namings N.

Fact 4. The analysis only depends on the free names and variables:

(i) If $me_1 =_P me_2$ and $(I, H, C, R) \models^l_{me_1} P$ then $(I, H, C, R) \models^l_{me_2} P$.
(ii) If $me_1 =_M me_2$ and $(I, H, C, R) \rhd_{me_1} M : \tilde{M}$ then $(I, H, C, R) \rhd_{me_2} M : \tilde{M}$.
(iii) If $me_1 =_N me_2$ and $(I, H, C, R) \Vdash_{me_1} N : \tilde{N}$ then $(I, H, C, R) \Vdash_{me_2} N : \tilde{N}$.

Proof. The proof of *(iii)* is immediate by inspection of the two cases for N. Next the proof of *(ii)* is straightforward by structural induction in M and using *(iii)*. Finally the proof *(i)* is by a straightforward structural induction using *(ii)* and *(iii)*; in particular, the induction hypothesis still applies in the cases of restriction, input of capabilities and input of names, where the naming environment is updated. □

Lemma 1. The analysis is invariant under the congruence:

If $P \equiv Q$ then $(I, H, C, R) \models^l_{me} P$ if and only if $(I, H, C, R) \models^l_{me} Q$.

Proof. The proof is by induction on the proof of $P \equiv Q$ and relies on Fact 4. Most cases are straightforward and we just illustrate one of the more interesting ones.

Consider the case $(\nu\, n^\mu)(P \mid Q) \equiv P \mid (\nu\, n^\mu)Q$ where $n \notin \underline{\mathrm{fn}}(P)$. It is immediate from the clauses of Table 4 that

$$(I, H, C, R) \models^l_{me} (\nu\, n^\mu)(P \mid Q)$$

is equivalent to $(I, H, C, R) \models^l_{me[n \mapsto \mu]} P \mid Q$ and hence to

$$(I, H, C, R) \models^l_{me[n \mapsto \mu]} P \ \wedge \ (I, H, C, R) \models^l_{me[n \mapsto \mu]} Q.$$

Since $n \notin \underline{\mathrm{fn}}(P)$ it follows from Fact 4 that this is equivalent to

$$(I, H, C, R) \models^l_{me} P \ \wedge \ (I, H, C, R) \models^l_{me[n \mapsto \mu]} Q$$

and using the clauses of Table 4 this is equivalent to

$$(I, H, C, R) \models^l_{me} P \mid (\nu\, n^\mu)Q$$

as desired. □

In order to establish the Subject Reduction Result we shall additionally need the following central result showing how substitutions work:

Lemma 2. The analysis enjoys the following substitution properties:

(i) If $(I, H, C, R) \rhd_{me} M : R^c(\beta^c)$ and $(I, H, C, R) \models^l_{me[x \mapsto \beta^c]} P$
then $(I, H, C, R) \models^l_{me} P[x \leftarrow M]$.

(ii) If $(I, H, C, R) \Vdash_{me} N : R^n(\beta^n)$ and $(I, H, C, R) \models^l_{me[u \mapsto \beta^n]} P$
then $(I, H, C, R) \models^l_{me} P[u \leftarrow N]$.

Proof. As a preparation for proving *(ii)* we first prove

> if $(I,H,C,R) \Vdash\!\models_{me} N : R^n(\beta^n)$ and $(I,H,C,R) \Vdash\!\models_{me[u \mapsto \beta^n]} N' : \tilde{N}'$
> then $(I,H,C,R) \Vdash\!\models_{me} N'[u \leftarrow N] : \tilde{N}'$

by inspection of the two cases for N'. Next we prove

> if $(I,H,C,R) \Vdash\!\models_{me} N : R^n(\beta^n)$ and $(I,H,C,R) \rhd_{me[u \mapsto \beta^n]} M : \tilde{M}$
> then $(I,H,C,R) \rhd_{me} M[u \leftarrow N] : \tilde{M}$

by structural induction in M and using the result just established; the proof is straightforward. We are then ready to prove *(ii)* by structural induction in P and using the two results just established; also this proof is straightforward.

As a preparation for proving *(i)* we first prove

> if $(I,H,C,R) \rhd_{me} M : R^c(\beta^c)$ and $(I,H,C,R) \rhd_{me[x \mapsto \beta^c]} M' : \tilde{M}'$ then
> $(I,H,C,R) \rhd_{me} M'[x \leftarrow M] : \tilde{M}'$

by structural induction in M'; the proof is straightforward. We are then ready to prove *(i)* by structural induction in P and using the result just established; also this proof is straightforward. □

Theorem 1. Subject Reduction:

> If $(I,H,C,R) \models^{I}_{me} P$ and $P \rightarrow^* Q$ then $(I,H,C,R) \models^{I}_{me} Q$.

Proof. The proof is by induction in the length of the derivation. In the induction step, $P \rightarrow^* S \rightarrow Q$, we have $(I,H,C,R) \models^{I}_{me} S$ from the induction hypothesis and need to show $(I,H,C,R) \models^{I}_{me} Q$.

We proceed by induction in the transition $S \rightarrow Q$. We first consider the reduction rules in the left-hand side of Table 3. The proof for the first three are straightforward using the induction hypothesis and the specification of Table 4; the proof for the fourth reduction rule, the one involving the congruence, is a direct consequence of the induction hypothesis and Lemma 1. We next turn our attention to the basic axioms in the right-hand side of Table 3. The proof for the in- and out-capabilities are straightforward using the specifications of Tables 4 and 5. In the case of the open-capability we additionally make use of Fact 3. Finally, in the case of communication we make use of Lemma 2. This concludes the proof. □

As a consequence, if (I,H,C,R) is an acceptable analysis estimate for the program $(me_\star, n^{l_\star}_\star[P_\star])$ of interest then it will continue being so for all the derivatives of the program.

Existence of analysis estimates. So far we have only shown how to check that a given estimate (I,H,C,R) is indeed an acceptable analysis estimate; we have not studied *(i)* whether or not acceptable analysis estimates always exist, and if they do, *(ii)* whether or not there always is a least analysis estimate.

To obtain these results we shall show that the set of acceptable analysis estimates constitutes a *Moore family* (sometimes called a model intersection property):

> A subset Y of a complete lattice $(L, \sqsubseteq)$ is a Moore family whenever $Y' \subseteq Y$ implies that $\sqcap Y' \in Y$.

By taking $Y' = \emptyset$ we see that a Moore family Y cannot be empty and by taking $Y' = Y$ we see that it always contains a least element; this will be essential for answering *(i)* and *(ii)* in the affirmative.

In our setting the complete lattice of interest is the set

$$\mathbf{InAmb} \times \mathbf{HNam} \times \mathbf{Comm} \times \mathbf{Env}$$

of tuples of mappings (I, H, C, R) and the ordering is the pointwise extension of the subset ordering. It follows that greatest lower bounds are calculated in a pointwise manner:

$$\sqcap_{j\in J}(I_j, H_j, C_j, R_j) = (\sqcap_{j\in J} I_j, \sqcap_{j\in J} H_j, \sqcap_{j\in J} C_j, \sqcap_{j\in J} R_j)$$

Since $I \sqsubseteq I'$ holds if and only if $I^{-1} \sqsubseteq I'^{-1}$ we also have $(\sqcap_{j\in J} I_j)^{-1} = \sqcap_{j\in J}(I_j^{-1})$. Finally, recall that $\sqcap_{j\in J}\cdots$ produces the greatest element $\top$ when J is the empty set.

We can now prove that the set of acceptable control flow estimates constitute a Moore family and hence that there always is a least estimate:

Theorem 2. The set $\{(I, H, C, R) \mid (I, H, C, R) \models^l_{me} P\}$ is a Moore family for all l, me and P.

Proof. The proof is in four parts. The first parts amounts to proving that

$$\{(I, H, C, R, \tilde{N}) \mid (I, H, C, R) \models_{me} N : \tilde{N}\}$$

is a Moore family for all me and N. This is immediate by inspection of the two cases for N.

The second part establishes that

$$\{(I, H, C, R) \mid (I, H, C, R) \models^l \tilde{m}\}$$

is a Moore family for all l and $\tilde{m}$. We proceed by inspection of the three cases for $\tilde{m}$. In the case of in-capabilities we assume that

$$\forall j \in J : (I_j, H_j, C_j, R_j) \models^l \mathsf{in}^{l^i} \tag{1}$$

and note that then $l^i \in I_j(l)$ for all $j \in J$ and hence $l^i \in (\sqcap_{j\in J} I_j)(l)$. Next consider l^a, μ, $l^{a'}$ and $l^{a''}$ such that $l^a \in (\sqcap_{j\in J} I_j)^{-1}(l^i)$, $\mu \in (\sqcap_{j\in J} H_j)(l^i)$, $l^{a'} \in (\sqcap_{j\in J} I_j)^{-1}(l^a)$, $l^{a''} \in (\sqcap_{j\in J} I_j)(l^{a'}) \cap (\sqcap_{j\in J} H_j)^{-1}(\mu) \cap \mathbf{Lab}^a$. It is immediate that we then have $l^a \in I_j^{-1}(l^i)$, $\mu \in H_j(l^i)$, $l^{a'} \in I_j^{-1}(l^a)$, $l^{a''} \in I_j(l^{a'}) \cap H_j^{-1}(\mu) \cap \mathbf{Lab}^a$ for all $j \in J$; it then follows from (1) that $l^a \in I_j(l^{a''})$ for all $j \in J$ and hence that $l^a \in (\sqcap_{j\in J} I_j)(l^{a''})$. In conclusion we then have

$$(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \models^l \mathsf{in}^{l^i}$$

as desired. The cases of out- and open-capabilities are similar.

The third part shows that

$$\{(I, H, C, R, \tilde{M}) \mid (I, H, C, R) \rhd_{me} M : \tilde{M}\}$$

is a Moore family for all M and me. The proof is by structural induction in M and let us consider the case $M_1.M_2$. Here we assume that

$$\forall j \in J : (I_j, H_j, C_j, R_j) \rhd_{me} M_1.M_2 : \tilde{M}_j$$

Hence there exist families $(\tilde{M}_{j1})_{j\in J}$ and $(\tilde{M}_{j2})_{j\in J}$ such that

$$\begin{aligned}
&\forall j \in J:\ (I_j, H_j, C_j, R_j) \rhd_{me} M_1 : \tilde{M}_{j1}\\
&\forall j \in J:\ (I_j, H_j, C_j, R_j) \rhd_{me} M_2 : \tilde{M}_{j2}\\
&\forall j \in J:\ \tilde{M}_j \supseteq \tilde{M}_{j1} \cup \tilde{M}_{j2}
\end{aligned}$$

By the induction hypothesis it follows that

$$\begin{aligned}
&(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \rhd_{me} M_1 : \cap_{j\in J}\tilde{M}_{j1}\\
&(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \rhd_{me} M_2 : \cap_{j\in J}\tilde{M}_{j2}\\
&(\cap_{j\in J}\tilde{M}_j) \supseteq (\cap_{j\in J}\tilde{M}_{j1}) \cup (\cap_{j\in J}\tilde{M}_{j2})
\end{aligned}$$

so that

$$(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \rhd_{me} M_1.M_2 : \cap_{j\in J}\tilde{M}_j$$

as desired. The remaining cases are similar.

Finally, the fourth part of the proof establishes the statement of the theorem. It is proved by structural induction in P and involves no new methods of reasoning beyond those already illustrated; we therefore only illustrate the case $N^{l^a}[P]$ of ambients. Here we assume that

$$\forall j \in J : (I_j, H_j, C_j, R_j) \models^{l}_{me} N^{l^a}[P]$$

It follows that there exists a family $(\tilde{N}_j)_{j\in J}$ such that

$$\begin{aligned}
&\forall j \in J:\ (I_j, H_j, C_j, R_j) \models^{l^a}_{me} P\\
&\forall j \in J:\ l^a \in I_j(l)\\
&\forall j \in J:\ (I_j, H_j, C_j, R_j) \Vdash_{me} N : \tilde{N}_j\\
&\forall j \in J:\ \tilde{N}_j \subseteq H_j(l^a)
\end{aligned}$$

Using the induction hypothesis and the results established above we then have

$$\begin{aligned}
&(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \models^{l^a}_{me} P\\
&l^a \in (\sqcap_{j\in J} I_j)(l)\\
&(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \Vdash_{me} N : \cap_{j\in J}\tilde{N}_j\\
&\cap_{j\in J}\tilde{N}_j \subseteq (\sqcap_{j\in J} H_j)(l^a)
\end{aligned}$$

and the desired

$$(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \models^{l}_{me} N^{l^a}[P]$$

then follows. This concludes the proof. □

3.3 Algorithmic properties

We now show how to compute in polynomial time the least solutions guaranteed by Theorem 2. We shall proceed in three stages. In the first we generate a set of master constraints. In the second we expand the master constraints to a larger set of conditional constraints. Finally, we solve the conditional constraints in polynomial time. As mentioned earlier we may without loss of generality assume that $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$ is finite.

Master constraints. To generate the master constraints we use four functions. The auxiliary functions **VM** and **VN** of Table 7 extract the "succinct results" produced by the analysis of capabilities and namings; these are the analogues of the sets $\tilde{M}$ and $\tilde{N}$ and here take the forms $\{\overline{m}_1, \cdots, \overline{m}_k\}$ and $\{\overline{n}\}$, where

$$\begin{array}{l}\overline{m} ::= \{\!|\mathsf{in}^{l^i}|\!\} \mid \{\!|\mathsf{out}^{l^o}|\!\} \mid \{\!|\mathsf{open}^{l^p}|\!\} \mid \mathtt{Rc}(\beta^c) \\ \overline{n} ::= \{\!|\mu|\!\} \mid \mathtt{Rn}(\beta^n)\end{array}$$

and $\beta^c \in \mathbf{Bnd}^c$, $\mu \in \mathbf{SNam}$ and $\beta^n \in \mathbf{Bnd}^n$. To make it clear that we are now talking about *syntax* we write $\{\!|\cdots|\!\}$ instead of $\{\cdots\}$ and $\mathtt{Rn}$ instead of R^n etc. Intuitively, the minimal $\tilde{M}$ such that $(I,H,C,R) \rhd_{me} M : \tilde{M}$ is given by $\bigcup\{\overline{m} \mid \overline{m} \in \mathbf{VM}[\![M]\!]_{me}\}$ and the minimal $\tilde{N}$ such that $(I,H,C,R) \Vdash_{me} N : \tilde{N}$ is given by $\bigcup\{\overline{n} \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\}$; we shall be slightly more precise in Lemma 3 below. (There is no need for an analogous function **VP** for processes since they do not have a "succinct" component.)

The constraint generation functions **CP** and **CM** of Tables 6 and 7 generate master constraints for processes and capabilities, respectively. Master constraints take the following rather permissive forms:

$\{\!|l|\!\} \subseteq \mathtt{I}(l^a)$	
$\{\!|\beta|\!\} \subseteq \mathtt{I}(l^a)$	
$\overline{n} \subseteq \mathtt{H}(l)$	
$\models^\bullet \mathsf{in}^l$	and similarly for out^l and open^l
$\forall l \in \mathbf{Lab}^i : \{\!|\mathsf{in}^l|\!\} \subseteq \mathtt{Rc}(\beta^c) \Rightarrow \{\!|l|\!\} \subseteq \mathtt{I}(l^a)$	and similarly for out^l and open^l
$\forall l \in \mathbf{Lab}^i : \{\!|\mathsf{in}^l|\!\} \subseteq \mathtt{Rc}(\beta^c) \Rightarrow \models^\bullet \mathsf{in}^l$	and similarly for out^l and open^l
$\forall l \in \mathtt{I}^{-1}(l^c) : \overline{m} \subseteq \mathtt{Cc}(l)$	and similarly for l^n, $\overline{n}$ and $\mathtt{Cn}$
$\forall l \in \mathtt{I}^{-1}(\beta^c) : \mathtt{Cc}(l) \subseteq \mathtt{Rc}(\beta^c)$	and similarly for β^n, $\mathtt{Cn}$ and $\mathtt{Rn}$

The constraint $\models^\bullet \mathsf{in}^{l^i}$ intuitively stands for $\models^l \mathsf{in}^{l^i}$ of Table 5 but without the condition $l^i \in I(l)$, i.e.

$$\begin{array}{c}\forall l^a \in I^{-1}(l^i) : \forall \mu \in H(l^i) : \forall l^{a'} \in I^{-1}(l^a) : \\ \forall l^{a''} \in I(l^{a'}) \cap H^{-1}(\mu) \cap \mathbf{Lab}^a : \\ l^a \in I(l^{a''})\end{array}$$

and similarly for $\models^\bullet \mathsf{out}^{l^o}$ and $\models^\bullet \mathsf{open}^{l^p}$. (There is no need for a constraint generation function **CN** for namings.)

We shall dispense with formally defining what it means for a control flow estimate (I,H,C,R) to satisfy a set of master constraints since the idea is clear: each constraint must be fulfilled when interpreting the formula with $\{\cdots\}$ for $\{\!|\cdots|\!\}$ and $R^n(\cdots)$ for $\mathtt{Rn}(\cdots)$ etc. We shall write $(I,H,C,R) \models \mathcal{C}$ when (I,H,C,R) satisfies a set $\mathcal{C}$ of constraints and $[\![\overline{m}]\!]_{(I,H,C,R)}$ for the value of $\overline{m}$ under the interpretation (I,H,C,R) and similarly for $[\![\overline{n}]\!]_{(I,H,C,R)}$. It is then straightforward to prove that the master constraints generated by Tables 6 and 7 faithfully model the acceptability relation of Tables 4 and 5.

Lemma 3. $(I,H,C,R) \models^l_{me} P$ if and only if $(I,H,C,R) \models \mathbf{CP}[\![P]\!]^l_{me}$.

Proof. The proof is by structural induction establishing also the following results:

$$(I,H,C,R) \Vdash_{me} N : \tilde{N} \text{ iff } \forall \overline{n} \in \mathbf{VN}[\![N]\!]_{me} : [\![\overline{n}]\!]_{(I,H,C,R)} \subseteq \tilde{N}$$

$$(I,H,C,R) \rhd_{me} M : \tilde{M} \text{ iff } \begin{array}{l}\forall \overline{m} \in \mathbf{VM}[\![M]\!]_{me} : [\![\overline{m}]\!]_{(I,H,C,R)} \subseteq \tilde{M} \wedge \\ (I,H,C,R) \models \mathbf{CM}[\![M]\!]_{me}\end{array}$$

We dispense with the details.

$$
\begin{array}{ll}
\mathbf{CP}[\![(\nu\, n^{\mu})P]\!]^{l}_{me} & = \mathbf{CP}[\![P]\!]^{l}_{me[n\mapsto\mu]} \\
\mathbf{CP}[\![0]\!]^{l}_{me} & = \emptyset \\
\mathbf{CP}[\![P \mid P']\!]^{l}_{me} & = \mathbf{CP}[\![P]\!]^{l}_{me} \cup \mathbf{CP}[\![P']\!]^{l}_{me} \\
\mathbf{CP}[\![!P]\!]^{l}_{me} & = \mathbf{CP}[\![P]\!]^{l}_{me} \\
\mathbf{CP}[\![N^{l^a}[P]]\!]^{l}_{me} & = \mathbf{CP}[\![P]\!]^{l^a}_{me} \cup \{\{\!|l^a|\!\} \subseteq \mathtt{I}(l)\} \cup \\
 & \quad \{\overline{n} \subseteq \mathtt{H}(l^a) \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\} \\
\mathbf{CP}[\![M.\, P]\!]^{l}_{me} & = \mathbf{CP}[\![P]\!]^{l}_{me} \cup \mathbf{CM}[\![M]\!]_{me} \cup \\
 & \quad \bigcup_{\overline{m}\in\mathbf{VM}[\![M]\!]_{me}} \left(\begin{array}{l}
\text{if } \overline{m} \text{ is } \{\!|\mathsf{in}^{l^i}|\!\} \text{ then } \{\models^{\bullet} \mathsf{in}^{l^i}, \{\!|l^i|\!\} \subseteq \mathtt{I}(l)\} \text{ else} \\
\text{if } \overline{m} \text{ is } \{\!|\mathsf{out}^{l^o}|\!\} \text{ then } \{\models^{\bullet} \mathsf{out}^{l^o}, \{\!|l^o|\!\} \subseteq \mathtt{I}(l)\} \text{ else} \\
\text{if } \overline{m} \text{ is } \{\!|\mathsf{open}^{l^p}|\!\} \text{ then } \{\models^{\bullet} \mathsf{open}^{l^p}, \{\!|l^p|\!\} \subseteq \mathtt{I}(l)\} \text{ else} \\
\left\{ \begin{array}{l}
\forall l^i \in \mathbf{Lab}^i : \{\!|\mathsf{in}^{l^i}|\!\} \subseteq \overline{m} \Rightarrow \models^{\bullet} \mathsf{in}^{l^i}, \\
\forall l^i \in \mathbf{Lab}^i : \{\!|\mathsf{in}^{l^i}|\!\} \subseteq \overline{m} \Rightarrow \{\!|l^i|\!\} \subseteq \mathtt{I}(l), \\
\forall l^o \in \mathbf{Lab}^o : \{\!|\mathsf{out}^{l^o}|\!\} \subseteq \overline{m} \Rightarrow \models^{\bullet} \mathsf{out}^{l^o}, \\
\forall l^o \in \mathbf{Lab}^o : \{\!|\mathsf{out}^{l^o}|\!\} \subseteq \overline{m} \Rightarrow \{\!|l^o|\!\} \subseteq \mathtt{I}(l), \\
\forall l^p \in \mathbf{Lab}^p : \{\!|\mathsf{open}^{l^p}|\!\} \subseteq \overline{m} \Rightarrow \models^{\bullet} \mathsf{open}^{l^p}, \\
\forall l^p \in \mathbf{Lab}^p : \{\!|\mathsf{open}^{l^p}|\!\} \subseteq \overline{m} \Rightarrow \{\!|l^p|\!\} \subseteq \mathtt{I}(l)
\end{array} \right\}
\end{array} \right) \\
\mathbf{CP}[\![\langle M \rangle^{l^c}]\!]^{l}_{me} & = \{\{\!|l^c|\!\} \subseteq \mathtt{I}(l)\} \cup \mathbf{CM}[\![M]\!]_{me} \cup \\
 & \quad \{\forall l^a \in \mathtt{I}^{-1}(l^c) : \overline{m} \subseteq \mathtt{Cc}(l^a) \mid \overline{m} \in \mathbf{VM}[\![M]\!]_{me}\} \\
\mathbf{CP}[\![\langle\!\langle N \rangle\!\rangle^{l^n}]\!]^{l}_{me} & = \{\{\!|l^n|\!\} \subseteq \mathtt{I}(l)\} \cup \\
 & \quad \{\forall l^a \in \mathtt{I}^{-1}(l^n) : \overline{n} \subseteq \mathtt{Cn}(l^a) \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\} \\
\mathbf{CP}[\![(x^{\beta^c}).\, P]\!]^{l}_{me} & = \mathbf{CP}[\![P]\!]^{l}_{me[x\mapsto\beta^c]} \cup \{\{\!|\beta^c|\!\} \subseteq \mathtt{I}(l)\} \cup \\
 & \quad \{\forall l^a \in \mathtt{I}^{-1}(\beta^c) : \mathtt{Cc}(l^a) \subseteq \mathtt{Rc}(\beta^c)\} \\
\mathbf{CP}[\![(\!(u^{\beta^n})\!).\, P]\!]^{l}_{me} & = \mathbf{CP}[\![P]\!]^{l}_{me[u\mapsto\beta^n]} \cup \{\{\!|\beta^n|\!\} \subseteq \mathtt{I}(l)\} \cup \\
 & \quad \{\forall l^a \in \mathtt{I}^{-1}(\beta^n) : \mathtt{Cn}(l^a) \subseteq \mathtt{Rn}(\beta^n)\}
\end{array}
$$

Table 6. Constraint generation (for processes).

Example 5. The set $\mathbf{CP}[\![\mathit{Firewall} \mid \mathit{Agent}]\!]^{l_\star}_{me_\star}$ of master contraints for the program $n^{l_\star}_{\star}[\mathit{Firewall} \mid \mathit{Agent}]$ of Example 1 consists of the following master constraints (assuming that $me_\star$ is an in Example 3 and ignoring the subprocesses P and Q):

$$
\begin{array}{lll}
\{\!|\mathrm{A}|\!\} \subseteq \mathtt{I}(l_\star) & & \{\!|w|\!\} \subseteq \mathtt{H}(\mathrm{A}) \\
\{\!|\mathrm{B}|\!\} \subseteq \mathtt{I}(\mathrm{A}) & & \{\!|k|\!\} \subseteq \mathtt{H}(\mathrm{B}) \\
\{\!|1|\!\} \subseteq \mathtt{I}(\mathrm{B}) & \models^{\bullet} \mathsf{out}^{1} & \{\!|w|\!\} \subseteq \mathtt{H}(1) \\
\{\!|2|\!\} \subseteq \mathtt{I}(\mathrm{B}) & \models^{\bullet} \mathsf{in}^{2} & \{\!|k'|\!\} \subseteq \mathtt{H}(2) \\
\{\!|3|\!\} \subseteq \mathtt{I}(\mathrm{B}) & \models^{\bullet} \mathsf{in}^{3} & \{\!|w|\!\} \subseteq \mathtt{H}(3) \\
\{\!|4|\!\} \subseteq \mathtt{I}(\mathrm{A}) & \models^{\bullet} \mathsf{open}^{4} & \{\!|k'|\!\} \subseteq \mathtt{H}(4) \\
\{\!|5|\!\} \subseteq \mathtt{I}(\mathrm{A}) & \models^{\bullet} \mathsf{open}^{5} & \{\!|k''|\!\} \subseteq \mathtt{H}(5) \\
\{\!|\mathrm{C}|\!\} \subseteq \mathtt{I}(l_\star) & & \{\!|k'|\!\} \subseteq \mathtt{H}(\mathrm{C}) \\
\{\!|6|\!\} \subseteq \mathtt{I}(\mathrm{C}) & \models^{\bullet} \mathsf{open}^{6} & \{\!|k|\!\} \subseteq \mathtt{H}(6) \\
\{\!|\mathrm{D}|\!\} \subseteq \mathtt{I}(\mathrm{C}) & & \{\!|k''|\!\} \subseteq \mathtt{H}(\mathrm{D})
\end{array}
$$

$$
\begin{array}{ll}
\mathbf{CM}[\![\mathsf{in}^{l^i} N]\!]_{me} & = \{\overline{n} \subseteq \mathtt{H}(l^i) \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\} \\
\mathbf{CM}[\![\mathsf{out}^{l^o} N]\!]_{me} & = \{\overline{n} \subseteq \mathtt{H}(l^o) \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\} \\
\mathbf{CM}[\![\mathsf{open}^{l^p} N]\!]_{me} & = \{\overline{n} \subseteq \mathtt{H}(l^p) \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\} \\
\mathbf{CM}[\![x]\!]_{me} & = \emptyset \\
\mathbf{CM}[\![\epsilon]\!]_{me} & = \emptyset \\
\mathbf{CM}[\![M_1.M_2]\!]_{me} & = \mathbf{CM}[\![M_1]\!]_{me} \cup \mathbf{CM}[\![M_2]\!]_{me} \\
\hline
\mathbf{VM}[\![\mathsf{in}^{l^i} N]\!]_{me} & = \{\{\!|\mathsf{in}^{l^i}|\!\}\} \\
\mathbf{VM}[\![\mathsf{out}^{l^o} N]\!]_{me} & = \{\{\!|\mathsf{out}^{l^o}|\!\}\} \\
\mathbf{VM}[\![\mathsf{open}^{l^p} N]\!]_{me} & = \{\{\!|\mathsf{open}^{l^p}|\!\}\} \\
\mathbf{VM}[\![x]\!]_{me} & = \{\mathtt{Rc}(me(x))\} \\
\mathbf{VM}[\![\epsilon]\!]_{me} & = \emptyset \\
\mathbf{VM}[\![M_1.M_2]\!]_{me} & = \mathbf{VM}[\![M_1]\!]_{me} \cup \mathbf{VM}[\![M_2]\!]_{me} \\
\hline
\mathbf{VN}[\![n]\!]_{me} & = \{\{\!|me(n)|\!\}\} \\
\mathbf{VN}[\![u]\!]_{me} & = \{\mathtt{Rn}(me(u))\}
\end{array}
$$

Table 7. Constraint generation (for capabilities and namings).

It is straightforward to check that the analysis estimate (I, H, C, R) of Example 3 indeed satisfies these master constraints. □

Suppose that the program is of size $p \geq 1$ and that the size of the finite set $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$ is $q \geq 1$. It is natural to assume that $q = O(p)$ as when $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$ is a finite set consisting only of the entities used in the program; however, we shall allow to let q be less than p so as to trade precision for efficiency.

Note that $\mathbf{VN}[\![N]\!]_{me}$ is always a singleton and that $\mathbf{CM}[\![M]\!]_{me}$ and $\mathbf{VM}[\![M]\!]_{me}$ contain no more elements than corresponds to the size of M. It is then immediate that constraint generation operates in time $O(p)$ and that it produces $O(p)$ master constraints each of size $O(1)$. — Since q may be much smaller than p it will be useful to observe that the number of constraints can also be given as $O(q^2)$; to see this simply note that each master constraint contains at most two "free" symbols from $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$. In this case constraint generation time should be estimated as $O(p + q^2)$.

Conditional constraints. The next stage is to expand the master constraints into sets of conditional constraints that do not involve quantifiers and that do not rely on the $\models^{\bullet} \cdots$ abbreviations adapted from Table 5. The general syntax of conditional constraints is based on constants (denoted *set*), variables (denoted *var*), conditions (denoted *cond*) and constraints (denoted *constr*):

$$
\begin{array}{rl}
\mathit{set} & ::= \{\!|l|\!\} \mid \{\!|\beta|\!\} \mid \{\!|\mu|\!\} \mid \{\!|\mathsf{in}^l|\!\} \mid \{\!|\mathsf{out}^l|\!\} \mid \{\!|\mathsf{open}^l|\!\} \\
\mathit{var} & ::= \mathtt{I}(l) \mid \mathtt{H}(l) \mid \mathtt{Cc}(l) \mid \mathtt{Cn}(l) \mid \mathtt{Rc}(\beta) \mid \mathtt{Rn}(\beta) \mid \cdots \\
\mathit{cond} & ::= \mathit{set} \subseteq \mathit{var} \\
\mathit{constr} & ::= \mathit{cond} \mid \mathit{cond} \Rightarrow \mathit{constr}
\end{array}
$$

To transform master constraints into conditional constraints we perform the following operations:

- expand $var_1 \subseteq var_2$ into $\forall set : set \subseteq var_1 \Rightarrow set \subseteq var_2$;
- unfold the definition of $\models^\bullet \cdots$;
- move quantifiers outermost;
- eliminate quantifiers by instantiating the bodies with all possible labels;
- eliminate all "-1" operations by changing $\{\!|x|\!\} \subseteq I^{-1}(y)$ to $\{\!|y|\!\} \subseteq I(x)$.

We illustrate the development for the following master constraint:

$$\forall l^i \in \mathbf{Lab}^i : \{\!|\mathsf{in}^{l^i}|\!\} \subseteq \mathtt{Rc}(\beta^c) \Rightarrow \models^\bullet \mathsf{in}^{l^i} \tag{2}$$

Straightforward application of the above operations gives rise to the following set of conditional constraints:

$$\left\{ \begin{array}{rcl|l} (\{\!|\mathsf{in}^{l^i}|\!\} \subseteq \mathtt{Rc}(\beta^c)) & \Rightarrow & & \\ (\{\!|l^i|\!\} \subseteq \mathtt{I}(l^a)) & \Rightarrow & & l^i \in \mathbf{Lab}^i, \\ (\{\!|\mu|\!\} \subseteq \mathtt{H}(l^i)) & \Rightarrow & & l^a \in \mathbf{Lab}^a, \\ (\{\!|l^a|\!\} \subseteq \mathtt{I}(l^{a'})) & \Rightarrow & & \mu \in \mathbf{SNam}, \\ (\{\!|l^{a''}|\!\} \subseteq \mathtt{I}(l^{a'})) & \Rightarrow & & l^{a'} \in \mathbf{Lab}^a, \\ (\{\!|\mu|\!\} \subseteq \mathtt{H}(l^{a''})) & \Rightarrow & & l^{a''} \in \mathbf{Lab}^a \\ \{\!|l^a|\!\} \subseteq \mathtt{I}(l^{a''}) & & & \end{array} \right\}$$

It turns out that this suffices for obtaining a polynomial time algorithm for computing the least solution; however, in the interest of obtaining a qubic time algorithm we shall "tile" the conditional constraints in the manner of [20]. To do so we introduce three auxiliary relations:

- $\{\!|l^i|\!\} \subseteq IN(\diamond)$ meaning that in^{l^i} has been found to be "active";
- $\{\!|l^{a''}|\!\} \subseteq NAM(l^i)$ meaning that l^i and $l^{a''}$ have the same name;
- $\{\!|l^a|\!\} \subseteq SIB(l^{a''})$ meaning that l^a and $l^{a''}$ are siblings.

(As will be shown below, their values are all derived from the estimate (I, H, C, R) under consideration.)

The master constraint (2) then gives rise to a set of "local" constraints and three sets of "global" constraints that are shared for all values of β^c occurring in (2). The set of "local" constraints, generated for each occurrence of (2), computes the IN relation from (I, H, C, R):

$$\left\{ (\{\!|\mathsf{in}^{l^i}|\!\} \subseteq \mathtt{Rc}(\beta^c)) \Rightarrow (\{\!|l^i|\!\} \subseteq IN(\diamond)) \,\middle|\, l^i \in \mathbf{Lab}^i \right\}$$

One of the sets of "global" constraints, shared for all occurrences of (2), computes the NAM relation from (I, H, C, R):

$$\left\{ \begin{array}{rcl|l} (\{\!|\mu|\!\} \subseteq \mathtt{H}(l^i)) & \Rightarrow & & l^i \in \mathbf{Lab}^i, \\ (\{\!|\mu|\!\} \subseteq \mathtt{H}(l^{a''})) & \Rightarrow & & \mu \in \mathbf{SNam}, \\ \{\!|l^{a''}|\!\} \subseteq NAM(l^i) & & & l^{a''} \in \mathbf{Lab}^a \end{array} \right\}$$

The other set of global constraints, shared for all occurrences of (2), computes the SIB relation from (I, H, C, R):

$$\left\{ \begin{array}{rcl|l} (\{\!|l^a|\!\} \subseteq \mathtt{I}(l^{a'})) & \Rightarrow & & l^a \in \mathbf{Lab}^a, \\ (\{\!|l^{a''}|\!\} \subseteq \mathtt{I}(l^{a'})) & \Rightarrow & & l^{a'} \in \mathbf{Lab}^a, \\ \{\!|l^a|\!\} \subseteq SIB(l^{a''}) & & & l^{a''} \in \mathbf{Lab}^a \end{array} \right\}$$

Type of Master Constraint	#Instances	#"Local"	#"Global"
$\{\!\{l\}\!\} \subseteq \mathtt{I}(l^a)$	$O(p)\,\&\,O(q^2)$	$O(1)$	$O(1)$
$\{\!\{\beta\}\!\} \subseteq \mathtt{I}(l^a)$	$O(p)\,\&\,O(q^2)$	$O(1)$	$O(1)$
$\overline{n} \subseteq \mathtt{H}(l)$	$O(p)\,\&\,O(q^2)$	$O(q)$	$O(1)$
$\models^\bullet \mathsf{in}^l$ etc.	$O(p)\,\&\,O(q)$	$O(1)$	$O(q^3)$
$\forall l \in \mathbf{Lab}^i : \{\!\{\mathsf{in}^l\}\!\} \subseteq \mathtt{Rc}(\beta^c) \Rightarrow \{\!\{l\}\!\} \subseteq \mathtt{I}(l^a)$ etc.	$O(p)\,\&\,O(q^2)$	$O(q)$	$O(1)$
$\forall l \in \mathbf{Lab}^i : \{\!\{\mathsf{in}^l\}\!\} \subseteq \mathtt{Rc}(\beta^c) \Rightarrow \models^\bullet \mathsf{in}^l$ etc.	$O(p)\,\&\,O(q)$	$O(q)$	$O(q^3)$
$\forall l \in \mathtt{I}^{-1}(l^c) : \overline{m} \subseteq \mathtt{Cc}(l)$ etc.	$O(p)\,\&\,O(q^2)$	$O(q)$	$O(q^3)$
$\forall l \in \mathtt{I}^{-1}(\beta^c) : \mathtt{Cc}(l) \subseteq \mathtt{Rc}(\beta^c)$ etc.	$O(p)\,\&\,O(q)$	$O(q^2)$	$O(1)$

Table 8. From master constraints to conditional constraints.

The final set of global constraints, shared for all occurrences of (2), performs the update of I:

$$\left\{ \begin{array}{rl|l} (\{\!\{l^i\}\!\} \subseteq IN(\diamond)) & \Rightarrow & \\ (\{\!\{l^i\}\!\} \subseteq \mathtt{I}(l^a)) & \Rightarrow & l^i \in \mathbf{Lab}^i, \\ (\{\!\{l^a\}\!\} \subseteq SIB(l^{a''})) & \Rightarrow & l^a \in \mathbf{Lab}^a, \\ (\{\!\{l^{a''}\}\!\} \subseteq NAM(l^i)) & \Rightarrow & l^{a''} \in \mathbf{Lab}^a \\ \{\!\{l^a\}\!\} \subseteq \mathtt{I}(l^{a''}) & & \end{array} \right\}$$

We proceed in a similar way for the other master constraints and write $\mathbf{C}[\![P]\!]^l_{me}$ for the set of conditional constraints obtained by expanding $\mathbf{CP}[\![P]\!]^l_{me}$. Once more we dispense with formally defining what it means for a control flow estimate (I, H, C, R) to satisfy a set of conditional constraints since the idea is clear: the auxiliary relations (IN, NAM, etc.) should be given as the least relations satisfying their defining constraints. In analogy with Lemma 3 we then have the following result:

Lemma 4. $(I, H, C, R) \models \mathbf{C}[\![P]\!]^l_{me}$ if and only if $(I, H, C, R) \models^l_{me} P$.

Table 8 shows for each form of master constraint how may instances are generated, how many "local" conditional constraints are generated by expanding each occurrence of a master constraint, and how many "global" conditional constraints are shared for all master constraints of the form considered. The number of master constraints is generally written as $O(p)\&O(q^2)$ to indicate that both bounds $O(p)$ and $O(q^2)$ apply as discussed previously; also note that only $O(p)\&O(q)$ master constraints involve $\models^\bullet \cdots$ since the relevant master constraints only contain one "free" symbol from $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$ and similarly for two other types of master constraints. In the example treated in detail, $O(q)$ "local" conditional constraints are produced for each occurrence of a master constraint and $O(q^3)$ "global" conditional constraints are shared; without the use of "tiling" [20] we would have generated $O(q^5)$ constraints for each occurrence of a master constraint.

Since $q = O(p)$ it follows that constraint generation operates in time $O(p^3)$ producing $O(p^3)$ conditional constraints of size $O(1)$. — It is useful to observe that constraint generation can also be estimated as $O(p + q^3)$ steps for generating $O(q^3)$ conditional constraints of size $O(1)$.

Constraint solving. We now consider how to turn $\mathbf{C}[\![P]\!]^l_{me}$ into the smallest acceptable analysis estimate (I, H, C, R). Perhaps the simplest approach is to use a Round Robin algorithm (see e.g. [17, Chapter 6]). A more efficient approach is based on worklist algorithms (see e.g. [17, Chapter 6]) and makes sure to consider constraints only when they are likely to have been enabled due to recent changes. This is the key

idea behind the approach of [11] and allows us to solve the conditional constraints in $\mathbf{C}[\![P]\!]^l_{me}$ in time proportional to their size.

Theorem 3 (Theorem 2 of [20]). The least control flow estimate,

$$\sqcap\{(I,H,C,R) \mid (I,H,C,R) \models^l_{me} P\},$$

can be computed in cubic time.

Proof. Thanks to Lemmas 3 and 4 we have

$$\sqcap\{(I,H,C,R) \mid (I,H,C,R) \models^l_{me} P\} = \sqcap\{(I,H,C,R) \mid (I,H,C,R) \models \mathbf{C}[\![P]\!]^l_{me}\}$$

and we already established that $\mathbf{C}[\![P]\!]^l_{me}$ has size $O(p^3)\&O(q^3)$ and can be generated in time $O(p^3)\&O(p+q^3)$. Thanks to the techniques of [11] the least solution can be found in time proportional to the size of the constraint system.

A constructive algorithm for computing the least solution can be found in Appendix A. Theorem 3 is attributed to [20] that developed "tiling" to establish a similar result for the communication-free fragment of the ambient calculus; no new complications were encountered when dealing with communication. □

The cubic time bound is satisfying since this is the complexity of control flow analyses also for functional and object oriented languages where control flow analysis is used with success also on "medium-sized" programs (between 10K and 100K lines of code). Furthermore, since q can be chosen arbitrarily small (although at the cost of precision) this gives confidence in exploring our current technology on fully realistic internet programs.

4 Validating Firewalls

In the examples we have studied a notion of firewall given by the passwords $\mathtt{k}$, $\mathtt{k}'$ and $\mathtt{k}''$ used for entering it. One aspect of being a firewall is that agents in the *approved* form must be allowed to enter. For the firewall proposed in Example 1 the approved form is $\mathtt{k}'^{\mathrm{C}}[\mathsf{open}^6\mathtt{k}.\,\mathtt{k}''^{\mathrm{D}}[Q]]$ and in Example 2 we showed that agents in this form can indeed enter the firewall: *Firewall* | *Agent* $\rightarrow^*$ $(\nu\,\mathtt{w}^{\mathrm{w}})\mathtt{w}^{\mathrm{A}}[P \mid Q]$ (assuming that $\mathtt{w} \notin \underline{\mathrm{fn}}(Q)$). It is shown in [7] that this can be strengthened to establish that *Firewall* | *Agent* is observationally equivalent to $(\nu\,\mathtt{w}^{\mathrm{w}})\mathtt{w}^{\mathrm{A}}[P \mid Q]$ (assuming that $\underline{\mathrm{fn}}(P) \cap \{\mathtt{k},\mathtt{k}',\mathtt{k}''\} = \emptyset = \underline{\mathrm{fn}}(Q) \cap \{\mathtt{w},\mathtt{k},\mathtt{k}',\mathtt{k}''\}$).

Another aspect of being a firewall, not dealt with in [7], is to ensure that processes not knowing the right passwords cannot enter. Due to the power of the ambient calculus this is not as trivial as it might appear at first sight. As an example, a process that does not initially know the passwords might nonetheless learn them by other means. As another example, the firewall might contain a trapdoor through which processes might be able to enter (see Example 6 below).

Intuitively, we define a process (or attacker) U to be *unaware* whenever $\underline{\mathrm{fn}}(U) \cap \{\mathtt{k},\mathtt{k}',\mathtt{k}''\} = \emptyset$. We then define a proposed firewall to be *protective* whenever the semantics of Section 2 prevents it from allowing any unaware process to enter.

Example 6. Consider the proposed firewall

$$\begin{aligned} \mathit{Firewall}' : (\nu\,\mathtt{w}^{\mathrm{w}})\mathtt{w}^{\mathrm{A}}[\mathtt{k}^{\mathrm{B}}[\mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k}'.\,\mathsf{in}^3\mathtt{w}] \mid \mathsf{open}^4\mathtt{k}'.\,\mathsf{open}^5\mathtt{k}''.P \\ \mid \mathtt{t}^{\mathrm{E}}[\mathsf{out}^7\mathtt{w}.\,\mathsf{in}^8\mathtt{w}.\,\mathsf{open}^9\mathtt{q}] \mid \mathsf{open}^{10}\mathtt{t}] \end{aligned}$$

that additionally contains a trapdoor $\mathtt{t}$. It is easy to check that

$$\mathit{Firewall}' \mid \mathit{Agent} \rightarrow^* (\nu\, \mathtt{w}^{w})\mathtt{w}^{\mathtt{A}}[\cdots \mid P \mid Q]$$

using *Agent* of Example 1 (assuming that $\mathtt{w} \notin \underline{\mathrm{fn}}(Q)$). But now the unaware process $\mathtt{q}^{\mathtt{F}}[\mathtt{in}^{11}\mathtt{t}.\, Q]$ can also enter as is shown by

$$\mathit{Firewall}' \mid \mathtt{q}^{\mathtt{F}}[\mathtt{in}^{11}\mathtt{t}.\, Q] \rightarrow^* (\nu\, \mathtt{w}^{w})\mathtt{w}^{\mathtt{A}}[\cdots \mid P \mid Q]$$

(again assuming that $\mathtt{w} \notin \underline{\mathrm{fn}}(Q)$) unlike what was intended. This means that *Firewall'* is not a protective firewall because it can be entered by a process not knowing the right passwords. □

In the development below we focus on one particular interface for firewalls, as given by the three passwords and formats shown above, but the development can be adapted to other interfaces as well. With respect to the chosen interface we then aim at developing a safe test for when a proposed realisation, e.g. *Firewall'* or *Firewall*, can be proved to be protective (i.e. to live up to the expectations).

We proceed by devising a test based on the control flow analysis; to facilitate this we need to formalise the notions of "proposed firewall" and "unaware process" (or unaware attacker) in the terminology of the control flow analysis. In essense this amounts to shifting our attention from names to stable names.

A *proposed firewall* is specified by a tuple of the form

$$(me_\star, ((\nu\, \mathtt{w}^{w})\mathtt{w}^{\mathtt{A}}[F]), k, k', k'')$$

such that $(me_\star, n_\star^{l_\star}[(\nu\, \mathtt{w}^{w})\mathtt{w}^{\mathtt{A}}[F]])$ is a program. As in the examples we assume that there are three passwords but the development can easily be adapted to an arbitrary selection of passwords.

To enable the proposed firewall to pass the test to be developed, it will be helpful to arrange that the naming environment respects the privacy of the name of the firewall, i.e. $me_\star^{-1}(w) = \emptyset$, that the naming environment respects the uniqueness of the passwords, i.e. $me_\star^{-1}(k) = \{\mathtt{k}\}$, $me_\star^{-1}(k') = \{\mathtt{k}'\}$ and $me_\star^{-1}(k'') = \{\mathtt{k}''\}$, and that the process F does not contain any of the stable names w, k, k' or k'', although this is not formally required.

For technical reasons we shall arrange that the the proposed firewall does not contain any distinguished symbols (in particular those marked "•") and that the naming environment does not map any names to the distinguished stable name $\mu_\bullet$; this can always be achieved by adjusting the choice of distinguished symbols and by Fact 2 this has no semantic consequences.

An *unaware attacker* (relative to the form of proposed firewall considered here) is a process U such that

- $(me_\star, n_\star^{l_\star}[U])$ is a program, and
- no free name in U is mapped to any of the "private" stable names, i.e. $\{me_\star(n) \mid n \in \underline{\mathrm{fn}}(U)\} \cap \{w, k, k', k''\} = \emptyset$.

When the above recommendations are adhered to, the second condition follows from the assumption that none of the passwords $\mathtt{k}$, $\mathtt{k}'$ or $\mathtt{k}''$ are free in U. Note that the first and second condition together establish the following stronger version of the second condition: $\{me_\star(n) \mid n \in \underline{\mathrm{fn}}(U)\} \cap \{\mu_\star, w, k, k', k''\} = \emptyset$; we shall exploit this fact in the proof of Proposition 1.

INPUT: a proposed firewall $(me_\star, ((\nu\, \mathtt{w}^{\mathtt{w}})\mathtt{w}^{\mathtt{A}}[F]), k, k', k'')$
without distinguished symbols in $\mathtt{w}^{\mathtt{A}}[F]$

OUTPUT: "accept" or "reject"

METHOD: let $\{n_1, \cdots, n_m\} = \{n \in dom(me_\star) \mid me_\star(n) \notin \{\mu_\star, w, k, k', k''\}\}$
let $n \notin dom(me_\star)$ and $x \in \mathbf{Var}^{\mathtt{c}}$ and $u \in \mathbf{Var}^{\mathtt{n}}$

$$\text{construct } U_\star^\bullet = \left(\begin{array}{l} \langle \mathsf{in}^{l^{\mathtt{i}}_\bullet} n\rangle^{l^{\mathtt{c}}_\bullet} \mid \langle \mathsf{out}^{l^{\mathtt{o}}_\bullet} n\rangle^{l^{\mathtt{c}}_\bullet} \mid \langle \mathsf{open}^{l^{\mathtt{p}}_\bullet} n\rangle^{l^{\mathtt{c}}_\bullet} \mid \\ (x^{\beta^{\mathtt{c}}_\bullet}).\, (\langle x\rangle^{l^{\mathtt{c}}_\bullet} \mid x.\,\mathbf{0}) \mid \\ \langle\!\langle n\rangle\!\rangle^{l^{\mathtt{n}}_\bullet} \mid \langle\!\langle n_1\rangle\!\rangle^{l^{\mathtt{n}}_\bullet} \mid \cdots \mid \langle\!\langle n_m\rangle\!\rangle^{l^{\mathtt{n}}_\bullet} \mid \\ (\langle\!\langle u^{\beta^{\mathtt{n}}_\bullet}\rangle\!\rangle).\, (\langle\!\langle u\rangle\!\rangle^{l^{\mathtt{n}}_\bullet} \mid u^{l^{\mathtt{a}}_\bullet}[\mathbf{0}] \mid \mathsf{in}^{l^{\mathtt{i}}_\bullet} u \mid \mathsf{out}^{l^{\mathtt{o}}_\bullet} u \mid \mathsf{open}^{l^{\mathtt{p}}_\bullet} u) \end{array}\right)$$

construct $U_\star = !(\nu n^{\mu_\bullet})(U_\star^\bullet \mid n^{l^{\mathtt{a}}_\bullet}[U_\star^\bullet])$
compute the least (I, H, C, R) such that
$(I, H, C, R) \models^{l_\star}_{me_\star} ((\nu\, \mathtt{w}^{\mathtt{w}})\mathtt{w}^{\mathtt{A}}[F]) \mid U_\star$
if $\exists l^a \in \mathbf{Lab}^a : l^a_\bullet \in I^+(l^a) \wedge w \in H(l^a)$
then "reject"
else "accept"

Table 9. Testing for protectiveness.

To develop a sound test for validating the protectiveness of a proposed firewall (see Table 9) we proceed in two stages. First recall that we defined a proposed firewall to be protective whenever the semantics prevents any unaware process from entering. The first stage of the development then is to define a related notion where the control flow analysis plays the role of the semantics: a firewall is *strongly protective* whenever the control flow analysis is able to demonstrate that no unaware process can enter the firewall. It follows from the correctness of the control flow analysis (Theorem 1) that a strongly protective firewall is also protective. Since the control flow analysis is approximate it would be unlikely for the converse result to hold; however, we shall see that *Firewall* is both protective and strongly protective whereas *Firewall'* is neither.

This then leads to the following interim suggestion for testing the strong protectiveness of a proposed firewall $(me_\star, ((\nu\, \mathtt{w}^{\mathtt{w}})\mathtt{w}^{\mathtt{A}}[F]), k, k', k'')$:

if there exists an unaware attacker U such that
for the least (I, H, C, R) satisfying $(I, H, C, R) \models^{l_\star}_{me_\star} ((\nu\, \mathtt{w}^{\mathtt{w}})\mathtt{w}^{\mathtt{A}}[F]) \mid U$
there exists $l^a_U \in \mathbf{Lab}^a$ labelling an entity in U and $l^a \in \mathbf{Lab}^a$
such that $l^a_U \in I^+(l^a) \wedge w \in H(l^a)$
then "reject" else "accept"

While this test is sound it involves a search over an infinity of unaware attackers and so is not readily implementable. The second stage of the development therefore is to restrict the search to a finite set of unaware attackers that are as hard to protect against as any other unaware attackers; in our case a single unaware attacker will do and it is called the *hardest attacker*. It amounts to the process $U_\star$ of Table 9. Clearly the hardest attacker should have access to all stable names except those of $\mu_\star, w, k, k', k''$; indeed, we ensure that only the stable name $\mu_\bullet$ is introduced internally. In a similar way we ensure that only the binders $\beta^{\mathtt{c}}_\bullet$ and $\beta^{\mathtt{n}}_\bullet$ and only the distinguished labels $l^{\mathtt{a}}_\bullet$, $l^{\mathtt{i}}_\bullet$, $l^{\mathtt{o}}_\bullet$, $l^{\mathtt{p}}_\bullet$, $l^{\mathtt{c}}_\bullet$ and $l^{\mathtt{n}}_\bullet$ are used internally. Over this universe of names, binders and labels, the hardest attacker must be able to perform all outputs

of names, to create ambients and capabilities with the required names, to output and input all kinds of capabilities and to enact all relevant capabilities.

It is hardly immediate that $U_\star$ of Table 9 lives up to these expectations. However, we are developing a sound test for strong protectiveness and so can restrict our attention to the level of granularity embodied in the control flow analysis. This allows us to prove in Proposition 1 below that the definition of $U_\star$ is sufficiently general to capture the behaviour of all unaware attackers — as far as the control flow analysis is concerned.

Example 7. To test *Firewall* from Example 1 and *Firewall'* from Example 6 we need to be more precise about the subprocess F; in our tests we have used

$$\mathsf{!p[in\ p \mid out\ p \mid open\ p \mid p[0]]}$$

(omitting labels) as an example of an unrestricted internal process. Then *Firewall* passes the test because $H^{-1}(w) = \{\mathrm{A}\}$ and $l^a_\bullet \notin I^+(\mathrm{A})$ but *Firewall'* fails the test because $H^{-1}(w) = \{\mathrm{A}\}$ and $l^a_\bullet \in I^+(\mathrm{A})$. □

As explained above the correctness of the test hinges on the following key result; it shows that, from the point of view of the analysis, it is as hard to protect a firewall against the process $U_\star$ of Table 9 as it is to protect it against any other unaware process U. The formulation uses the operation $\lfloor U \rfloor$ also used to express Fact 2: all stable names, binders and labels are replaced by the appropriate distinguished stable names, binders and labels.

Proposition 1. Let $(me_\star, ((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]), k, k', k'')$ be a proposed firewall as demanded in Table 9 and let (I, H, C, R) be as in Table 9. Then

$$(I, H, C, R) \models^{l_\star}_{me_\star} ((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid \lfloor U \rfloor$$

whenever U is an unaware attacker.

Proof. The proof is in two parts. The first draws a number of consequences from the fact that $(I, H, C, R) \models^{l_\star}_{me_\star} U_\star$ and the second proves a number of "general" results from which $(I, H, C, R) \models^{l_\star}_{me_\star} \lfloor U \rfloor$ immediately follows.

Part 1. For the first part we unfold the formula defining $(I, H, C, R) \models^{l_\star}_{me_\star} U_\star$. Since $U^\bullet_\star$ appears outermost as well as inside $n^{l^a_\bullet}[\cdots]$ we get:

$$\begin{aligned} I(l_\star) &\supseteq \{l^a_\bullet, l^i_\bullet, l^o_\bullet, l^p_\bullet, l^c_\bullet, l^n_\bullet, \beta^c_\bullet, \beta^n_\bullet\} \\ I(l^a_\bullet) &\supseteq \{l^a_\bullet, l^i_\bullet, l^o_\bullet, l^p_\bullet, l^c_\bullet, l^n_\bullet, \beta^c_\bullet, \beta^n_\bullet\} \end{aligned} \tag{3}$$

In the first two lines of $U^\bullet_\star$ we output all three kinds of capabilities and then input them again; similarly in the last two lines we output all the stable names $\mu_\bullet, me_\star(n_1), \cdots, me_\star(n_m)$ accessible to $U^\bullet_\star$ and then input them again; this yields:

$$\begin{aligned} R^c(\beta^c_\bullet) &\supseteq \{\mathsf{in}^{l^i_\bullet}, \mathsf{out}^{l^o_\bullet}, \mathsf{open}^{l^p_\bullet}\} \\ R^n(\beta^n_\bullet) &\supseteq \{\mu_\bullet, me_\star(n_1), \cdots, me_\star(n_m)\} \end{aligned} \tag{4}$$

More generally, the fact that we input what has been output, ensures that the clauses for input of Table 4 establish the following containments:

$$\begin{aligned} &\forall l^a \in I^{-1}(\beta^c_\bullet) : C^c(l^a) \subseteq R^c(\beta^c_\bullet) \\ &\forall l^a \in I^{-1}(\beta^n_\bullet) : C^n(l^a) \subseteq R^n(\beta^n_\bullet) \end{aligned} \tag{5}$$

In a similar way we also make sure to output what has just been input and the clauses for output of Table 4 then establish the "dual" containments:

$$\begin{array}{l} \forall l^a \in I^{-1}(l^c_\bullet) : C^c(l^a) \supseteq R^c(\beta^c_\bullet) \\ \forall l^a \in I^{-1}(l^n_\bullet) : C^n(l^a) \supseteq R^n(\beta^n_\bullet) \end{array} \tag{6}$$

In the second line of $U^\bullet_\star$ we perform the capability just input; since this takes place both outermost and inside $n^{l^a_\bullet}[\cdots]$, the clause for capabilities gives:

$$\forall \tilde{m} \in R^c(\beta^c_\bullet) : \forall l \in \{l_\star, l^a_\bullet\} : (I,H,C,R) \models^l \tilde{m} \tag{7}$$

since the $\tilde{M}$ used for validating $x.\mathbf{0}$ must satisfy $\tilde{M} \supseteq R^c(\beta^c_\bullet)$. Similarly, in the fourth line of $U^\bullet_\star$ we construct ambients and capabilities with the names just input; this yields:

$$\begin{array}{l} R^n(\beta^n_\bullet) \subseteq H(l^a_\bullet) \\ R^n(\beta^n_\bullet) \subseteq H(l^i_\bullet) \\ R^n(\beta^n_\bullet) \subseteq H(l^o_\bullet) \\ R^n(\beta^n_\bullet) \subseteq H(l^p_\bullet) \end{array} \tag{8}$$

since the $\tilde{N}$ used for validating $u^{l^a_\bullet}[\mathbf{0}]$ must satisfy $\tilde{N} \supseteq R^n(\beta^n_\bullet)$ and similarly for the three capabilities.

Part 2. We now consider the second part of the proof. We shall say that a naming environment me is acceptable for the process P whenever it defines all names and variables in P, i.e. $\underline{\mathrm{fv}}(P) \cup \underline{\mathrm{fn}}(P) \subseteq dom(me)$, and whenever it only maps variables and names to "acceptable" symbols:

$$range(me) \subseteq \{\mu_\bullet, me_\star(n_1), \cdots, me_\star(n_m), \beta^c_\bullet, \beta^n_\bullet\}.$$

In a similar way we define the acceptability of me for a capability M and a naming N.

First we prove for all me and N that

$$\begin{array}{l} (I,H,C,R) \Vdash_{me} N : R^n(\beta^n_\bullet) \\ \text{provided } me \text{ is acceptable for } N \end{array} \tag{9}$$

where (I,H,C,R) is as above. The proof is by inspection of the two cases for N. The case n is immediate given (4) and the acceptability of me. The case u is trivial.

Next we prove for all me and M that

$$\begin{array}{l} (I,H,C,R) \rhd_{me} \lfloor M \rfloor : R^c(\beta^c_\bullet) \\ \text{provided } me \text{ is acceptable for } M \end{array} \tag{10}$$

where again (I,H,C,R) is as above. The proof is by structural induction in $\lfloor M \rfloor$. In the cases $\mathsf{in}^{l^i_\bullet} N$, $\mathsf{out}^{l^o_\bullet} N$ and $\mathsf{open}^{l^p_\bullet} N$ we take $\tilde{N} = R^n(\beta^n_\bullet)$; the result is then immediate from (9), (4) and (8). The cases x and ϵ are trivial and the case $M_1.M_2$ is immediate from the induction hypothesis.

Finally we prove for all me and P that

$$\begin{array}{l} \forall l \in \{l_\star, l^a_\bullet\} : (I,H,C,R) \models^l_{me} \lfloor P \rfloor \\ \text{whenever } me \text{ is acceptable for } P \end{array} \tag{11}$$

and where (I,H,C,R) once more is as above. The proof is by structural induction in $\lfloor P \rfloor$. The case $(\nu\, n^{\mu_\bullet})\lfloor P \rfloor$ follows from the induction hypothesis (since $me[n \mapsto \mu_\bullet]$

is acceptable for $\lfloor P \rfloor$). The cases $\mathbf{0}$, $\lfloor P \rfloor \mid \lfloor P' \rfloor$ and $!\lfloor P \rfloor$ are immediate using the induction hypothesis. In the case $N^{l^a_\bullet}[\lfloor P \rfloor]$ we choose $\tilde{N} = R^n(\beta^n_\bullet)$; the result then is a consequence of the induction hypothesis, (3), (9) and (8). In the case $\lfloor M \rfloor.\lfloor P \rfloor$ we take $\tilde{M} = R^c(\beta^c_\bullet)$; the result then follows from the induction hypothesis, (10) and (7). In the case for $\langle M \rangle^{l^c_\bullet}$ we take $\tilde{M} = R^c(\beta^c_\bullet)$; the result then is a consequence of (3), (10) and (6). In the case for $\langle\!\langle N \rangle\!\rangle^{l^n_\bullet}$ we take $\tilde{N} = R^n(\beta^n_\bullet)$; the result then follows from (3), (9) and (6). The case $(x^{\beta^c_\bullet}).\lfloor P \rfloor$ is a consequence of the induction hypothesis (since $me[x \mapsto \beta^c_\bullet]$ is acceptable), (3) and (5). The case $(\!(u^{\beta^n_\bullet})\!).\lfloor P \rfloor$ follows from the induction hypothesis (since $me[u \mapsto \beta^n_\bullet]$ is acceptable), (3) and (5).

Returning to the unaware process U we observe that

$$\{me_\star(n) \mid n \in \underline{\mathrm{fn}}(U)\} \subseteq \{me_\star(n_1), \cdots, me_\star(n_m)\}$$

and it then follows from (11) and Fact 4 that $me_\star$ can be restricted so as to be acceptable for U; this gives $(I, H, C, R) \models^{l_\star}_{me_\star} \lfloor U \rfloor$ as desired. □

When $(\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]$ passes the test and U is an unaware process we want to show that no subambient of U ever passes inside $\mathtt{w}$. Informally, this will take the form of assuming that $((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid U \rightarrow^* S$ and guaranteeing that S contains no subambient $\mathtt{w}^{l^a_1}[\cdots \mathtt{u}^{l^a_2}[\cdots]\cdots]$ where $\mathtt{u}$ comes from U. To formalise this we shall avail ourselves of Fact 2 that allows us to arrange the labelling to suit our needs. Indeed, if $((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid U \rightarrow^* S$ then $((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid \lfloor U \rfloor \rightarrow^* S'$ for some S' such that $\lfloor S \rfloor = \lfloor S' \rfloor$.

Theorem 4. Suppose that $(\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]$ passes the test of Table 9 and that U is an unaware process; if $((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid \lfloor U \rfloor \rightarrow^* S$ then S contains no subterm $n_1^{l^a_1}[\cdots n_2^{l^a_2}[\cdots]\cdots]$ where n_1 has stable name w and l^a_2 is $l^a_\bullet$.

Proof. Letting (I, H, C, R) be as in Table 9, it follows from Proposition 1 that $(I, H, C, R) \models^{l_\star}_{me_\star} ((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid \lfloor U \rfloor$. Then $(I, H, C, R) \models^{l_\star}_{me_\star} S$ follows from Theorem 1. Suppose next, for the sake of contradiction, that S does contain a subprocess $n_1^{l^a_1}[\cdots n_2^{l^a_2}[\cdots]\cdots]$ where n_1 has stable name w and l^a_2 is $l^a_\bullet$. Then it follows from $(I, H, C, R) \models^{l_\star}_{me_\star} S$ that $l^a_\bullet \in I^+(l^a_1) \wedge w \in H(l^a_1)$ showing that the test could not have been passed. □

Theorem 5. The test in Table 9 operates in cubic time.

Proof. Given a firewall of size $O(p)$ we may without loss of generality assume that the set $\{n_1, \cdots, n_m\}$ of variables constructed in Table 9 does not contain any variables not in the firewall; this ensures that also $((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid U_\star$ will have size $O(p)$. It follows from Theorem 3 that the least solution can be found in time $O(p + q^3)$ where, as before, q is the number of elements of $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$. This dominates the $O(q^3)$ operations needed to calculate the transitive closure I^+ of the relation I. Hence the overall operation of Table 9 is $O(p^3) \& O(p + q^3)$ where p is the size of the firewall and $q = O(p)$ can be chosen "arbitrarily" small. □

In summary, we have succeeded in using the control flow analysis to devise a cubic time algorithm for correctly validating that a proposed firewall is indeed protective; unlike [18] this development applies to the full ambient calculus. By judicious choice of the sets of labels, binders and markers the test can be performed in near-linear time (although at the cost of precision); this gives confidence in exploring our current approach on fully realistic internet programs.

5 Conclusion

Static analysis provides a summary of the behaviour of programs; we have shown that classical control flow analysis techniques can be adapted to tackle the much more dynamic setting of the ambient calculus. Our flow logic approach facilitates fully automatic validation of an analysis estimate as well as fully automatic computation of the best estimate; in spirit this is rather similar to the approaches of type inference. Type systems have already been extensively used to study the properties of web-based languages and related calculi (e.g. [1, 8]); the use of flow logic offers a flexible approach to adapting the vast amount of more "traditional" approaches to static analysis [17].

In this paper we developed a control flow analysis for the ambient calculus building on recent developments for the π-calculus [2–4] and the ambient calculus [14, 18, 19, 22]. In fact a notion of cryptography is implicitly part of the ambient calculus as presented here. As in the spi-calculus it is the restriction operator that is used to model "secrets" that cannot be guessed by brute force attack. Thus a message M, encrypted under the key K, is represented simply as the ambient $K[M]$ whereas an attempt at decrypting such a message is represented by the ambient $\mathsf{open}K$. If K is a secret only known to the principals $P_1, \cdots, P_n$ the entire system is represented as $(\nu K)(P_1 \mid \cdots \mid P_n)$ where each P_i may contain $K[M]$ as well as $\mathsf{open}K$.

More importantly we demonstrated how a careful exploitation of the detailed operation of the control flow analysis allowed us to construct an attacker that was as hard to protect against as any other attacker; this is somewhat reminiscent of the identification of hard problems in a given complexity class. This allowed us to predict the operation of the firewall in conjunction with all unaware attackers based on its operation in conjunction with the hardest attacker; if it successfully protects against the hard attacker it will also protect against all other attackers not knowing the required passwords. We believe this to be typical of applications where software developed by subcontractors is *validated* before being embedded in the software system under construction. To make this practical we ensured that the test could be performed in cubic time.

It is important to stress that we circumvent the undecidability of dealing with all possible execution contexts, in particular all attackers, by coarsening the "level of granularity" of our observations to coincide with those of the static analysis. We maintain soundness because we proved the static analysis to be sound (but of course not complete) with respect to the dynamic semantics.

The analysis leading to the hardest attacker can be compared with the approaches of the "Dolev-Yao tradition" [10], including [5, 23, 12]. Indeed, the analyses performed in these studies amount to an "informal" analysis of the capabilities of the attacker, leading to an inductively defined behaviour. Such an analysis is not straightforward when the computational capabilities are modified, such as when adding mobility. The advantage of our approach therefore is two-fold. On the one hand, it allows us to make the analysis in a "formal" manner, because we can prove theorems with respect to the dynamic semantics (as mitigated by the analysis). On the other hand, the techniques used to implement the analysis provide insights on how to define the capabilities of the hardest attacker.

In summary, we have illustrated a novel approach to the validation of safety and security properties of software systems. We are hopeful that it will scale up to other calculi and web-based languages with explicit cryptographic primitives. By considering flow logics that express more powerful analyses it is likely that one can capture attackers in a more precise manner; perhaps one can show that the

firewall protects againsts attackers knowing only some of the secret passwords or even against attackers knowing all the secret passwords but unable to use them in the appropriate manner (as might be the case for an agent representing a "courier" service).

Acknowledgements. We would like to thank Jacob Grydholt Jensen for working with us on [18] and Helmut Seidl for cooperating on "tiling" [20]. We would also like to thank the referees for useful feed-back (leading to an improved complexity result).

References

1. M. Abadi. Secrecy by typing in security protocols. In *Proceedings of Theoretical Aspects of Computer Software 1997, LNCS 1281*, pages 611–638. Springer, 1997.
2. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis for the π-calculus. In *Proceedings of CONCUR'98, LNCS 1466*, pages 84–98. Springer, 1998.
3. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis of processes for no read-up and no write-down. In *Proceedings of* 2^{nd} *FoSSaCS'99, LNCS 1578*, pages 120–134. Springer, 1999.
4. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for the π-calculus with applications to security. *Information and Computation*, (to appear), 2000.
5. D. Bolignano. An approach to the formal verification of cryptographic protocols. In *Proc. 3rd ACM Conf. on Computer and Communications Security*, pages 106–118. ACM Press, 1996.
6. L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for mobile ambients. In *Proceedings of ICALP'99, LNCS 1644*, pages 230–239. Springer, 1999.
7. L. Cardelli and A. D. Gordon. Mobile ambients. In *Proceedings of FoSSaCS'98, LNCS 1378*, pages 140–155. Springer, 1998.
8. L. Cardelli and A. D. Gordon. Types for mobile ambients. In *Proceedings of POPL'99*, pages 79–92. ACM Press, 1999.
9. L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proceedings of POPL'00*, pages 365–377. ACM Press, 2000.
10. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, 1983.
11. W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.
12. F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand Spaces: Why is a security protocol correct? In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Press, 1998.
13. Li Gong. *Inside Java 2 Platform Security.* Addison-Wesley, 1999.
14. R. R. Hansen, J. G. Jensen, F. Nielson, and H. Riis Nielson. Abstract interpretation of mobile ambients. In *Proceedings of SAS'99, LNCS 1694*, pages 135–148, 1999.
15. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (I and II). *Information and Computation*, 100(1):1–77, 1992.
16. F. Nielson and H. Riis Nielson. Flow logics and operational semantics. *Electronic Notes of Theoretical Computer Science*, 10, 1998.
17. F. Nielson, H. Riis Nielson, and C. Hankin. *Principles of Program Analysis.* Springer, 1999.
18. F. Nielson, H. Riis Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in mobile ambients. In *Proceedings of Concur'99, LNCS 1664*, pages 463–477. Springer, 1999.
19. F. Nielson, H. Riis Nielson, and M. Sagiv. A Kleene analysis of mobile ambients. In *Proceedings of ESOP'00, LNCS 1782*, pages 305–319. Springer, 2000.
20. F. Nielson and H. Seidl. Control-flow analysis in cubic time. In *Proceedings of ESOP'2001, LNCS (to appear)*. Springer, 2001.
21. H. Riis Nielson and F. Nielson. Flow logics for constraint based analysis. In *Proc. CC'98, LNCS 1383*, pages 109–127. Springer, 1998.

GIVEN: a bounded number of binary relations $R_1, \cdots, R_m$,
and a bound on the size of each conditional constraint.

INPUT: a set C of conditional constraints over the binary relations.

OUTPUT: the least solution $(\mathtt{R}_1, \cdots, \mathtt{R}_m)$ such that $(\mathtt{R}_1, \cdots, \mathtt{R}_m) \models C$.

INITIALISE:
```
for i := 1 to m do
  for each argument arg occurring in C do
    R_i(arg) := ∅;

for each set ⊆ var occurring in C do
  INFL[set,var] := NIL;

cno := 0; stack := NIL;
for each set_k ⊆ var_k ⇒ ··· set_1 ⊆ var_1 ⇒ set_0 ⊆ var_0 in C do
  cno:= cno + 1;
  CT[cno] := (set_k ⊆ var_k ⇒ ··· set_1 ⊆ var_1 ⇒ set_0 ⊆ var_0);
  if k = 0 then push((set_0, var_0)) else
    for i := 1 to k do
      INFL[set_i,var_i] := CONS(cno, INFL[set_i,var_i]);
```

ITERATE:
```
while stack ≠ NIL do
  let (set,var) = pop() in
    for cno in INFL[set,var] do
      let (set_k ⊆ var_k ⇒ ··· set_1 ⊆ var_1 ⇒ set_0 ⊆ var_0) = CT[cno] in
        if ⋀_{i=1}^{k} [set_i] ⊆ [var_i] then push((set_0, var_0)) else skip;
```

USING:
```
procedure push((set,var)) is
if [set] ⊆ [var] then skip else
   [var] := [var] ∪ [set]; stack := CONS((set,var),stack);

function pop() is
let (set,var) = HEAD(stack) in
  stack := TAIL(stack); return((set,var));
```

Table 10. Worklist solution of constraints.

22. H. Riis Nielson and F. Nielson. Shape analysis for mobile ambients. In *Proceedings of POPL'00*, pages 142–154. ACM Press, 2000.
23. L. C. Paulson. Proving properties of security protocols by induction. In *Proc. 10th Computer Security Foundations Workshop*, pages 70–83. IEEE, 1997.

A Constraint Solving

We now consider how to turn a set C of conditional constraints of size $|C|$ into the smallest acceptable analysis estimate. To this end we develop a worklist algorithm (see e.g. [17, Chapter 6]). It operates on data structures $\mathtt{R}_1, \cdots, \mathtt{R}_m$ corresponding to a constant number of binary relations. Additionally, it makes use of a list `stack` of "bit positions" that have just been set to 1 and whose consequences remain to be explored, a table `CT` that maps constraint numbers to the constraint in question, and a table `INFL` that for each "bit position" gives a list of constraint numbers influenced by that "bit position". We operate on lists using the constructors `CONS` and `NIL` and the destructors `HEAD` and `TAIL`. To operate on `stack` we make use of the function push for setting a "bit position" to 1 and for placing the "bit position" on `stack`, and of the function pop for returning the topmost "bit position" on `stack` and at the same time removing it from `stack`.

The worklist algorithm is displayed in Table 10. The initialisation of $\mathtt{R}_1, \cdots, \mathtt{R}_m$ amounts to setting $(\mathtt{R}_1, \cdots, \mathtt{R}_m)$ equal to the least element $(\bot, \cdots, \bot)$ of the appropriate lattice of values. The initialisation next sets `stack` and all `INFL[`set, var`]` to `NIL` and then computes the correct contents of the structures `stack`, `CT` and `INFL` by iterating through all conditional constraints in $\mathcal{C}$. The iteration phase amounts to propagating "bit positions" as long as there are new "bit positions" recently set to 1 whose consequences need to be explored. We write $[\![set]\!]$ and $[\![variable]\!]$ whenever we interpret the entities *set* and *var* as denoting values and variables, i.e. whenever they are used for comparisons or for updating the data structures. The algorithm clearly terminates because no "bit position" is placed twice on `stack` thanks to the test performed by the `push` operation.

We can now state the correctness of the worklist algorithm:

Lemma 5. Table 10 computes $\sqcap\{(R_1, \cdots, R_m) \mid (R_1, \cdots, R_m) \models \mathcal{C}\}$.

Proof. Write $(\bot, \cdots, \bot)$ for the least element and write

$$(R_1^\bullet, \cdots, R_m^\bullet) = \sqcap\{(R_1, \cdots, R_m) \mid (R_1, \cdots, R_m) \models \mathcal{C}\}$$

for the least solution (guaranteed by the obvious extension of Theorem 2). It follows from the "Horn clause" format of the conditional constraints that the invariant

$$(\bot, \cdots, \bot) \sqsubseteq (\mathtt{R}_1, \cdots, \mathtt{R}_m) \sqsubseteq (R_1^\bullet, \cdots, R_m^\bullet)$$

remains fulfilled throughout the iteration. Next write $(\mathtt{R}_1, \cdots, \mathtt{R}_m)$ for the resulting value. It follows from the above that

$$(\mathtt{R}_1, \cdots, \mathtt{R}_m) \sqsubseteq (R_1^\bullet, \cdots, R_m^\bullet)$$

and since it is clear that all constraints in $\mathcal{C}$ are fulfilled upon termination it follows that

$$(\mathtt{R}_1, \cdots, \mathtt{R}_m) \models \mathcal{C}.$$

Using the definition of $(R_1^\bullet, \cdots, R_m^\bullet)$ we then have

$$(\mathtt{R}_1, \cdots, \mathtt{R}_m) = \sqcap\{(R_1, \cdots, R_m) \mid (R_1, \cdots, R_m) \models \mathcal{C}\}$$

showing that the algorithm computes the least solution. □

To state the complexity of the worklist algorithm we assume that there is a constant bound on the size of the conditional constraints occurring in $\mathcal{C}$:

Lemma 6 (Special case of [11]). Table 10 operates in time $O(|\mathcal{C}|)$.

Proof. The initialisation of $\mathtt{R}_1, \cdots, \mathtt{R}_m$ takes time $O(|\mathcal{C}|)$. Setting `INFL[`set, var`]` to `NIL` also takes time $O(|\mathcal{C}|)$. The remaining initialisation performs $O(1)$ steps for each of the $O(|\mathcal{C}|)$ conditional constraints (recalling that $k = O(1)$, i.e. the length of conditions is bounded by a constant). For the iteration at most $O(|\mathcal{C}|)$ "bit positions" are placed on the `stack` thanks to the test performed by the `push` operation and to the fact that a "bit position" set to 1 is never reset to 0 given the "Horn clause" format of the conditional constraints. Ignoring the inspection of `INFL[`set, var`]` we therefore use time $O(|\mathcal{C}|)$. The inspection of `INFL[`set, var`]` can be amortised over all iterations since each constraint is only considered $O(1)$ times (again because $k = O(1)$) and thus takes overall time $O(|\mathcal{C}|)$. □

Revocation and Tracing Schemes for Stateless Receivers

Dalit Naor* Moni Naor† Jeff Lotspiech*

February 24, 2001

Abstract

We deal with the problem of a center sending a message to a group of users such that some subset of the users is considered revoked and should not be able to obtain the content of the message. We concentrate on the *stateless receiver* case, where the users do not (necessarily) update their state from session to session. We present a framework called the *Subset-Cover* framework, which abstracts a variety of revocation schemes including some previously known ones. We provide sufficient conditions that guarantees the security of a revocation algorithm in this class.

We describe two explicit Subset-Cover revocation algorithms; these algorithms are very flexible and work for any number of revoked users. The schemes require storage at the receiver of $\log N$ and $\frac{1}{2}\log^2 N$ keys respectively (N is the total number of users), and in order to revoke r users the required message lengths are of $r \log N$ and $2r$ keys respectively. We also provide a general *traitor tracing* mechanism that can be integrated with any Subset-Cover revocation scheme that satisfies a "bifurcation property". This mechanism does not need an a priori bound on the number of traitors and does not expand the message length by much compared to the revocation of the same set of traitors.

The main improvements of these methods over previously suggested methods, when adopted to the stateless scenario, are: (1) reducing the message length to $O(r)$ *regardless* of the coalition size while maintaining a single decryption at the user's end (2) provide a seamless integration between the revocation and tracing so that the tracing mechanisms does not require any change to the revocation algorithm.

1 Introduction

The problem of a Center transmitting data to a large group of receivers so that only a predefined subset is able to decrypt the data is at the heart of a growing number of applications. Among them are pay-TV applications, multicast communication, secure distribution of copyright-protected material (e.g. music) and audio streaming. The area of Broadcast Encryption deals with methods to efficiently broadcast information to a dynamically changing group of users who are allowed to receive the data. It is often convenient to think of it as a Revocation Scheme, which addresses the case where some subset of the users are excluded from receiving the information. In such scenarios it is also desirable to have a Tracing Mechanism, which enables the efficient tracing of leakage, specifically, the source of keys used by illegal devices, such as pirate decoders or clones.

One special case is when the receivers are stateless. In such a scenario, a receiver is not capable of recording the past history of transmissions and change its state accordingly. Instead, its operation must be

*IBM Almaden Research Center. Email: {*lots, dalit*}*@almaden.ibm.com*

†Department of Computer Science and Applied Math, Weizmann Institute. Work done while visiting IBM Almaden Research Center and Stanford University. Email: *naor@wisdom.weizmann.ac.il*

based on the current transmission and its initial configuration. Stateless receivers are important for the case where the receiver is a device that is not constantly on-line, such as a media player (e.g. a CD or DVD player where the "transmission" is the current disc), a satellite receiver (GPS) and perhaps in multicast applications.

This paper introduces very efficient revocation schemes which are specifically suitable for stateless receivers. Our approach is quite general. We define a framework of such algorithms, called Subset-Cover algorithms, and provide a sufficient condition for an algorithm in this family to be secure. Furthermore, we suggest two particular implementations for schemes in this family; the performance of the second method is substantially better than any previously known algorithm for this problem (see Section 1.1). We also provide a general property ('bifurcation') of revocation algorithms in our framework that allows efficient tracing methods, *without* modifying the underlying revocation scheme.

Copyright Protection

An important application that motivates the study of revocation and tracing mechanisms is Copyright Protection. The distribution of copyright protected content for (possibly) disconnected operations involves encryption of the content on a media. The media (such as CD, DVD or a flash memory card) typically contains in its header the encryption of the key K which encrypts the content following the header. Compliant devices, or receivers, store appropriate decryption keys that can be used to decrypt the header and in turn decrypt the content. A *copyright protection mechanism* defines the algorithm which assigns keys to devices and encrypts the content.

An essential requirement from a copyright protection mechanism is the ability to revoke, during the lifetime of the system, devices that are being used illegally. It is expected that some devices will be compromised, either via reverse engineering or due to sloppy manufacturing of the devices. As a result keys of a number of compromised devices can then be cloned to form a decrypting unit. This copyright protection violation can be combated by revoking the keys of the compromised devices. Note that devices are stateless as they are assumed to have no capability of dynamically storing any information (other than the original information that is stored at the time of manufacturing) and also since they are typically not connected to the world (except via the media). Hence, it is the responsibility of the media to carry the current state of the system at the time of recording in terms of revoked devices.

It is also highly desirable that the revocation algorithm be coupled with a traitors tracing mechanism. Specifically, a well-designed copyright protection mechanism should be able to combat piracy in the form of illegal boxes or clone decoding programs. Such decoders typically contain the identities of a *number* of devices that are cooperating; furthermore, they are hard to disassemble[1]. The tracing mechanism should therefore treat the illicit decoder as a black box and simply examine its input/output relationship. A combination of a revocation and a tracing mechanism provides a powerful tool for combating piracy: finding the identities of compromised devices, revoking them and rendering the illegal boxes useless.

Caveat. The goal of a copyright protection mechanism is to create a legal channel of distribution of content and to disallow its abuse. As a consequence, an illegal distribution will require the establishment of alternative channels and should not be able to piggyback on the legitimate channel[2]. Such alternative channels should be combated using other means and is not under the scope of the techniques developed in this paper, thought techniques such as revocation may be a useful deterrent against rough users.

[1]For instance, the software clone known as DeCSS, that cracked the DVD Video "encryption", is shielded by a tamper-resistant software tool which makes it very hard to reverse engineer its code and know its details such as receivers identities or its decoding strategy.

[2]For instance in the case of cable TV the pirates should be forced to create their own cable network.

1.1 Related Work

Broadcast Encryption. The area of Broadcast Encryption was first formally studied (and coined) by Fiat and Naor in [23] and has received much attention since then. To the best of our knowledge the scenario of stateless receivers has not been considered explicitly in the past in a scientific paper, but in principle any scheme that works for the connected mode may be converted to a scheme for stateless receivers. (Such conversion may require including with any transmission the entire 'history' of revocation events.) When discussing previously proposed schemes we will consider their performance when adapted to the stateless receiver scenario.

To survey previous results we should fix our notation. Let N be the total number of users in the system let r be the size of the revoked set $\mathcal{R}$. Another important parameter that is often considered is t, the upper bound on the size of the coalition an adversary can assemble. The algorithms in this paper do not require such a bound and we can think of $t = r$; on the other hand some previously proposed schemes depend on t but are independent of r. The Broadcast Encryption method of [23] is one such scheme which allows the removal of any number of users as long as at most t of them collude. There the message length is $O(t \log^2 t)$, a user must store a number of keys that is logarithmic in t and the amount of work required by the user is $\tilde{O}(r/t)$ decryptions.

The logical-tree-hierarchy (LKH) scheme, suggested independently by Wallner et. al. [42] and Wong et. al. [43], is designed for the connected mode for multicast re-keying applications. It revokes a single user at a time, and updates the keys of all remaining users. If used in our scenario, it requires a transmission of $2r \log N$ keys to revoke r users, each user should store $\log N$ keys and the amount of work each user should do is $r \log N$ encryptions (the expected number is $O(r)$ for an average user). These bounds are somewhat improved in [12, 13, 32], but unless the storage at the user is extremely high they still require a transmission of length $\Omega(r \log N)$. The key assignment of this scheme and the key assignment of our first method are similar; see further discussion on comparing the two methods in Section 3.1.

Luby and Staddon [31] considered the information theoretic setting and devised bounds for any revocation algorithms under this setting. Their "Or Protocol" fits our Subset-Cover framework. In fact, as we show in Section 3.3, our second algorithm (the Subset Difference method) which is *not* information theoretic, beats their lower bound (Theorem 12 in [31]). Garay, Staddon and Wool [27] introduced the notion of *long-lived broadcast encryption*. In this scenario, keys of compromised decoders are no longer used for encryptions. The question they address is how to adapt the broadcast encryption scheme so as to maintain the security of the system for the good users.

The method of Kumar et. al [30] enables one-time revocation of up to r users with message lengths of $O(r \log N)$ and $O(r^2)$.

CPRM [18] is one of the methods that explicitly considers the stateless scenario. Our Subset Difference method outperforms CPRM by a factor of about 25 in the number of revocations it can handle when all the other parameters are fixed; Section 4.5 contains a detailed description and comparison.

Tracing Mechanisms. The notion of a tracing system was first introduced by Chor, Fiat and Naor in [16], and was later refined to the Threshold Traitor model in [34], [17]. Its goal is to distribute decryption keys to the users so as to allow the detection of at least one key that is used in a pirate box or clone using keys of at most t users. *Black-box tracing* assumes that only the outcome of the decoding box can be examined. [34], [17] provide combinatorial and probabilistic constructions that guarantee tracing with high probability. To trace t traitors, they require each user to store $O(t \log N)$ keys and to perform a single decryption operation. The message length is $4t$. The public key tracing scheme of Boneh and Franklin [7] provides a number-theoretic deterministic method for tracing. Note that in all of the above methods t is an *a-priori* bound.

Preventing Leakage of Keys. The problem of preventing illegal leakage of keys has been attacked by a

number of quite different approaches. The legal approach, suggested by Pfitzmann [39], requires a method that not only traces the leaker but also yields a proof for the liability of the traitor (the user whose keys are used by an illegal decoder). Hence, leakage can be fought via legal channels by presenting this proof to a third party. The self enforcement approach, suggested by Dwork, Lotspiech and Naor [20], aims at *deterring* users from revealing their personal keys. The idea is to provide a user with personal keys that contain some sensitive information about the user which the user will be reluctant to disclose. The trace-and-revoke approach is to design a method that can trace the identity of the user whose key was leaked; in turn, this user's key is revoked from the system for future uses. The results in this paper fall into the latter category, albeit in a slightly relaxed manner. Although our methods assure that leaked keys will become useless in future transmissions, it may not reveal the actual *identities* of all leaking keys, thus somewhat lacking self-enforcement.

Content Tracing: In addition to tracing leakers who give away their private keys there are methods that attempt to detect illegal users who redistribute the content *after* it is decoded. This requires the assumption that good watermarking techniques with the following properties are available: it is possible to insert one of several types of watermarks into the content so that the adversary cannot create a "clean" version with no watermarks (or a watermark it did not receive). Typically, content is divided into segments that are watermarked separately. This setting with protection against collusions was first investigated by Boneh and Shaw [9]. A related setting with slightly stronger assumptions on the underlying watermarking technique was investigated in [24, 5, 41]. By introducing the *time* dimension, Fiat and Tassa [24] propose the dynamic tracing scenario in which the watermarking of a segment depends on feedback from the previous segment and which detects all traitors. Their algorithm was improved by Berkman, Parnas and Sgall [5], and a scheme which requires no real-time computation/feedback for this model was given in Safavani-Naini and Wang [41]. Content tracing is relevant to our scenario in that any content tracing mechanism can be combined with a key-revocation method to ensure that the traced users are indeed revoked and do not receive new content in the future. Moreover, the tracing methods of [24] are related to the tracing *algorithm* of Section 5.2.

Integration of tracing and revocation. Broadcast encryption can be combined with tracing schemes to yield trace-and-revoke schemes. While Gafni et. al. [26] only consider combinatorial constructions, the schemes in Naor and Pinkas [35] are more general. The previously best known trace-and-revoke algorithm of [35] can tolerate a coalition of at most t users. It requires to store $O(t)$ keys at each user and to perform $O(r)$ decryptions; the message length is r keys, however these keys are elements in a group where the Decisional Diffie-Hellman problem is difficult and therefore these keys may be longer than symmetric keys. The tracing model of [35] is not a "pure" black-box model. Anzai et. al [2] employs a similar method for revocation, but without tracing capabilities. Our results improve upon this algorithm both in the work that must be performed at the user and in the lengths of the keys transmitted in the message.

1.2 Summary of Results

In this paper we define a generic framework encapsulating several previously proposed revocation methods, called *Subset-Cover* algorithms. These algorithms are based on the principle of covering all non-revoked users by disjoint subsets from a predefined collection. We define the security of a revocation scheme and provide a sufficient condition (key-indistinguishability) for a revocation algorithm in the Subset-Cover Framework to be secure. An important consequence of this framework is the separation between long-lived keys and short-term keys. The framework can be easily extended to the public-key scenario.

We then provide two new instantiations of revocation schemes in the Subset-Cover Framework, with a

Method	Message Length	Storage @ Receiver	Processing time	no. Decryptions
Complete Subtree	$r \log \frac{N}{r}$	$\log N$	$O(\log \log N)$	1
Subset Difference	$2r-1$	$\frac{1}{2}\log^2 N$	$O(\log N)$	1

Figure 1: Performance Tradeoff for the Complete Subtree method and the Subset Difference method

different performance tradeoff (summarized in Table 1.2[3]). Both instantiations are tree-based, namely the subsets are derived from a virtual tree structure imposed on all devices in the system. The first requires a message length of $r \log N$ and storage of $\log N$ keys at the receiver and constitutes a moderate improvement over previously proposed schemes; the second exhibits a substantial improvement: it requires a message length of $2r-1$ (in the worst case, or $1.38r$ in the average case) and storage of $\frac{1}{2}\log^2 N$ keys at the receiver. N is the total number of devices, and r is the number of *revoked* devices. Furthermore, these algorithms are r-flexible, namely they do not assume an upper bound of the number of revoked receivers.

Thirdly, we present a tracing mechanism that works in tandem with a Subset-Cover revocation scheme. We identify the *bifurcation property* for a Subset-Cover scheme. Our two constructions of revocation schemes posses this property. We show that every scheme that satisfies the bifurcation property can be combined with the tracing mechanism to yield a trace-and-revoke scheme. The integration of the two mechanisms is seamless in the sense that no change is required for any one of them. Moreover, no a-priori bound on the number of traitors is needed for our tracing scheme. In order to trace t illegal users, the first revocation method requires a message length of $t \log N$, and the second revocation method requires a message length of $5t$.

Main Contributions: the main improvements that our methods achieve over previously suggested methods, when adopted to the stateless scenario, are:

- Reducing the message length to linear in r regardless of the coalition size, while maintaining a single decryption at the user's end. This applies also to the case where public keys are used, *without* a substantial length increase.
- The seamless integration between revocation and tracing: the tracing mechanism does not require any change of the revocation algorithm and no a priori bound on the number of traitors, even when all traitors cooperate among themselves.
- The rigorous treatment of the security of such schemes, identifying the effect of parameter choice on the overall security of the scheme.

Organization of the paper. Section 2 describes the framework for Subset-Cover algorithms. Section 3 describes two specific implementations of such algorithms. Section 4 presents extensions, implementation issues, public-key methods, application to multicasting as well as casting the recently proposed CPRM method (for DVD-audio and SD cards) in the Subset-Cover Framework. Section 5 provides the traitors tracing extensions to Subset-Cover revocation algorithms and their seamless integration. In Section 6 we define the "key-indistinguishability" property and provide the main theorem characterizing the security of revocation algorithms in the Subset-Cover framework.

[3]Note that the comparison in the processing time between the two methods treats an application of a pseudo-random generator and a lookup operation as having the same cost, even though they might be quite different. More explicitly, the processing of both methods consists of $O(\log \log N)$ lookups; in addition, the Subset Difference method requires at most $\log N$ applications of a pseudo-random generator.

2 The Subset-Cover Revocation Framework

2.1 Preliminaries - Problem Definition

Let $\mathcal{N}$ be the set of all users, $|\mathcal{N}| = N$, and $\mathcal{R} \subset \mathcal{N}$ be a group of $|\mathcal{R}| = r$ users whose decryption privileges should be revoked. The goal of a revocation algorithm is to allow a center to transmit a message M to all users such that any user $u \in \mathcal{N} \setminus \mathcal{R}$ can decrypt the message correctly, while even a coalition consisting of all members of $\mathcal{R}$ cannot decrypt it. The exact definition of the latter is provided in Section 6.

A system consists of three parts: (1) An initiation scheme, which is a method for assigning the receivers secret information that will allow them to decrypt. (2) The broadcast algorithm - given a message M and the set $\mathcal{R}$ of users that should be revoked outputs a ciphertext message M' that is broadcast to all receivers. (3) A decryption algorithm - a (non-revoked) user that receives ciphertext M' using its secret information should produce the original message M. Since the receivers are stateless, the output of the decryption should be based on the current message and the secret information only.

2.2 The Framework

We present a framework for algorithms which we call *Subset-Cover*. In this framework an algorithm defines a collection of subsets $S_1, \ldots, S_w$, $S_j \subseteq \mathcal{N}$. Each subset S_j is assigned (perhaps implicitly) a long-lived key L_j; each member u of S_j should be able to deduce L_j from its secret information. Given a revoked set $\mathcal{R}$, the remaining users $\mathcal{N} \setminus \mathcal{R}$ are partitioned into disjoint subsets $S_{i_1}, \ldots, S_{i_m}$ so that

$$\mathcal{N} \setminus \mathcal{R} = \bigcup_{j=1}^{m} S_{i_j}$$

and a session key K is encrypted m times with $L_{i_1}, \ldots, L_{i_m}$.

Specifically, an algorithm in the framework uses two encryption schemes:

- A method $F_K : \{0,1\}^* \mapsto \{0,1\}^*$ to encrypt the message itself. The key K used will be chosen fresh for each message M - a session key - as a random bit string. F_K should be a fast method and should not expand the plaintext. The simplest implementation is to Xor the message M with a stream cipher generated by K.
- A method to deliver the session key to the receivers, for which we will employ an encryption scheme. The keys L here are long-lived. The simplest implementation is to make $E_L : \{0,1\}^\ell \mapsto \{0,1\}^\ell$ a block cipher.

An exact discussion of security requirements of these primitives is given in Section 6. Some suggestions for the implementation of F_K and E_L are given in Section 4.1. The algorithm consists of three components:

Scheme Initiation : Every receiver u is assigned private information I_u. For all $1 \leq i \leq w$ such that $u \in S_i$, I_u allows u to deduce the key L_i corresponding to the set S_i. Note that the keys L_i can be chosen either (i) uniformly at random and independently from each other (which we call the *information-theoretic* case) or (ii) as a function of other (secret) information (which we call the computational case), and thus may not be independent of each other.

The **Broadcast algorithm** at the Center:

1. Choose a session encryption key K.

2. Given a set $\mathcal{R}$ of revoked receivers, the center finds a partition of the users in $\mathcal{N} - \mathcal{R}$ into disjoint subsets $S_{i_1}, \ldots, S_{i_m}$. Let $L_{i_1}, \ldots, L_{i_m}$ be the keys associated with the above subsets.

3. The center encrypts K with keys $L_{i_1}, \ldots, L_{i_m}$ and sends the ciphertext

$$\langle [i_1, i_2, \ldots, i_m, E_{L_{i_1}}(K), E_{L_{i_2}}(K), \ldots, E_{L_{i_m}}(K)], F_K(M) \rangle$$

The portion in square brackets preceding $F_K(M)$ is called the *header* and $F_K(M)$ is called the *body*.

The **Decryption step** at the receiver u, upon receiving a broadcast message

$$\langle [i_1, i_2, \ldots, i_m, C_1, C_2, \ldots, C_m], M' \rangle :$$

1. Find i_j such that $u \in S_{i_j}$ (in case $u \in \mathcal{R}$ the result is **null**).

2. Extract the corresponding key L_{i_j} from I_u.

3. Compute $D_{L_{i_j}}(C_j))$ to obtain K.

4. Compute $D_K(M')$ to obtain and output M.

A particular implementation of such scheme is specified by (1) the collection of subsets $S_1, \ldots, S_w$ (2) the key assignment to each subset in the collection (3) a method to cover the non-revoked receivers $\mathcal{N} \setminus \mathcal{R}$ by disjoint subsets from this collection, and (4) A method that allows each user u to find its cover S and compute its key L_S from I_u. The algorithm is evaluated based upon three parameters:

i. Message Length - the length of the header that is attached to $F_K(M)$, which is proportional to m, the number of sets in the partition covering $\mathcal{N} \setminus \mathcal{R}$.

ii. Storage size at the receiver - how much private information (typically, keys) does a receiver need to store. For instance, I_u could simply consists of all the keys S_i such that $u \in S_i$, or if the key assignment is more sophisticated it should allow the computation of all such keys.

iii. Message processing time at receiver. We often distinguish between decryption and other types of operations.

It is important to characterize the dependence of the above three parameters in both N and r. Specifically, we say that a revocation scheme is *flexible with respect to r* if the storage at the receiver is not a function of r. Note that the efficiency of setting up the scheme and computing the partition (given $\mathcal{R}$) is not taken into account in the algorithm's analysis. However, for all schemes presented in this paper the computational requirements of the sender are rather modest: finding the partition takes time linear in $|\mathcal{R}| \log \mathcal{N}$ and the encryption is proportional to the number of subsets in the partition.

2.3 Security of the Framework: Key-Indistinguishability

Section 6 discusses in detail the security of an algorithm in the Subset-Cover framework. Intuitively, we identify a critical property that is required from the key-assignment method in order to provide a secure Subset-Cover algorithm. We say that a subset-cover algorithm satisfies the "key-indistinguishability" property if for every subset S_i its key L_i is indistinguishable from a random key given all the information of all users that are *not* in S_i. Note that any scheme in which the keys to all subsets are chosen independently satisfies this property. We then proceed to show that any subset-cover algorithm that satisfies the key-indistinguishability property provides a secure encryption of the message (the overall encryption security is expressed as a function of the security provided by E and F).

3 Two Subset-Cover Revocation Algorithms

We now describe two instantiations of revocation schemes in the Subset-Cover framework with a different performance tradeoff, as summarized in table 1.2. Each is defined over a different collection of subsets. Both schemes are r-flexible, namely they work with any number of revocations. In the first scheme, the key assignment is information-theoretic and in the other scheme the key assignment is computational. The first method is natural; the second method is more involved, and exhibits a substantial improvement over previous methods.

In both schemes the subsets and the partitions are obtained by imagining the receivers as the leaves in a rooted full binary tree with N leaves (assume that N is a power of 2). Such a tree contains $2N-1$ nodes (leaves plus internal nodes) and for any $1 \leq i \leq 2N-1$ we assume that v_i is a node in the tree. The systems differ in the collections of subsets they consider.

3.1 The Complete Subtree Method

The collection of subsets $S_1, \ldots, S_w$ in our first scheme corresponds to all complete subtrees in the full binary tree with N leaves. For any node v_i in the full binary tree (either an internal node or a leaf, $2N-1$ altogether) let the subset S_i be the collection of receivers u that correspond to the leaves of the subtree rooted at node v_i. In other words, $u \in S_i$ iff v_i is an ancestor of u. The key assignment method is simple: assign an independent and random key L_i to every node v_i in the complete tree. Provide every receiver u with the $\log N + 1$ keys associated with the nodes along the path from the root to leaf u.

For a given set $\mathcal{R}$ of revoked receivers, let $u_1, \ldots, u_r$ be the leaves corresponding to the elements in $\mathcal{R}$. The method to partition $\mathcal{N} \setminus \mathcal{R}$ into disjoint subsets is as follows. Consider the (directed) Steiner Tree $ST(\mathcal{R})$ defined by the set $\mathcal{R}$ of vertices and the root, i.e. the minimal subtree of the full binary tree that connects all the leaves in $\mathcal{R}$. $ST(\mathcal{R})$ is unique. Let $S_{i_1}, \ldots, S_{i_m}$ be all the subtrees of the original tree that "hang" off $ST(\mathcal{R})$, that is, all subtrees whose roots $v_1, \ldots, v_m$ are adjacent to nodes of outdegree 1 in $ST(\mathcal{R})$, but they are not in $ST(\mathcal{R})$. The next claim follows immediately and shows that this construction is indeed a cover, as required.

Claim 1 *1. Every leaf $u \notin \mathcal{R}$ is in exactly one subset in the above partition.*

2. A leaf $u \in \mathcal{R}$ does not belong to any subset in the above partition.

The cover size: The Steiner tree $ST(\mathcal{R})$ has r leaves. An internal node is in $ST(\mathcal{R})$ iff it is on some path to a point in $\mathcal{R}$, therefore there are at most $r \log N$ nodes in $ST(\mathcal{R})$. A finer analysis takes into account double counting of the nodes closer to the root and the fact that a node of outdegree 2 in $ST(\mathcal{R})$ does not produce a subset, and shows that the number of subsets is at most $r \log(N/r)$. The analysis is as follows: note that the number of sets is exactly the number of degree 1 nodes in $ST(\mathcal{R})$. Assume by induction on the tree height that this is true for trees of depth i, i.e. that in a subtree with r leaves the maximum number of nodes of degree 1 is at most $r \cdot (i - \log r)$. Then consider a tree of depth $i+1$. If all the leaves are contained in one subtree of depth i, then by induction the total number of nodes of degree 1 is at most $r \cdot (i - \log r) + 1 \leq r \cdot (i + 1 - \log r)$. Otherwise, the number of nodes of degree 1 is the number of nodes of degree 1 in the left subtree (that has $r_1 \geq 1$ leaves) plus the number of nodes of degree 1 in the right subtree (that has $r_2 \geq 1$ leaves) and $r = r_1 + r_2$. By induction, this is at most $r_1 \cdot (i - \log r_1) + r_2 \cdot (i - \log r_2) = r \cdot i - (r_1 \log r_1 + r_2 \log r_2) \leq r \cdot (i + 1 - \log r)$ since $(r_1 \log r_1 + r_2 \log r_2) \geq r(\log r - 1)$. Note that this is also the average number of subsets (where the r leaves are chosen at random).

The Decryption Step: Given a message

$$\langle [i_1, \ldots, i_m, E_{L_{i_1}}(K), E_{L_{i_2}}(K), \ldots, E_{L_{i_m}}(K)], F_K(M)] \rangle$$

a receiver u needs to find whether any of its ancestors is among $i_1, i_2, \ldots i_m$; note that there can be only one such ancestor, so u may belong to at most one subset.

There are several ways to facilitate an efficient search in this list[4]. First consider a generic method that works whenever each receiver is a member of relatively few subsets S_i: the values $i_1, i_2, \ldots i_m$ are put in a hash table and in addition a *perfect hash function* h of the list is transmitted as well (see [15] for a recent survey of such functions). The length of the description of h can be relatively small compared to the length of the list i.e. it can be $o(m \log w)$. The receiver u should check for all i such that $u \in S_i$ whether i is in the list by computing $h(i)$. In our case this would mean checking $\log N$ values.

Furthermore, suppose that we are interested in using as few bits as possible to represent the collection of subsets used $\{i_1, i_2, \ldots i_m\}$. The information-theoretic bound on the number of bits needed is $\lceil \log \binom{w}{m} \rceil$, which is roughly $m \log w/m$, using Stirling's approximation. (Note that when $m \approx \sqrt{w}$ this represents a factor 2 compression compared to storing $\{i_1, i_2, \ldots i_m\}$ explicitly.) However we are interested in a succinct representation of the collection that allows efficient lookup in this list. It turns out that with an *additive* factor of $O(m + \log \log w)$ bits it is possible to support an $O(1)$ lookup, see [10, 38]; the results they provide are even slightly better, but this bound is relatively simple to achieve.

It turns out that we can do even better for the complete subtree method, given the special structure. For each node u, the desired ancestor i_j in the list is the one with which u and i_j have the longest common prefix. Searching for this can be done by $\log \log N$ comparisons given the right preprocessing of the data, see [33].

Summarizing, in the complete subtree method (i) the message header consists of at most $r \log \frac{N}{r}$ keys and encryptions (ii) receivers have to store $\log N$ keys and (iii) processing a message requires $O(\log \log N)$ operations plus a *single* decryption operation.

Security: The key assignment in this method is information theoretic, that is keys are assigned randomly and independently. Hence the "key-indistinguishability" property of this method follows from the fact that no $u \in \mathcal{R}$ is contained in any of the subsets $i_1, i_2, \ldots i_m$, as stated in Claim 1.

Theorem 2 *The Complete Subtree Revocation method requires (i) message length of at most* $r \log \frac{N}{r}$ *keys (ii) to store* $\log N$ *keys at a receiver and (iii)* $O(\log \log N)$ *operations plus a* single *decryption operation to decrypt a message. Moreover, the method is secure in the sense of Definition 11.*

Comparison to the Logical Key Hierarchy (LKH) approach : Readers familiar with the LKH method of [42, 43] may find it instructive to compare it to the Complete Subtree Scheme. The main similarity lies in the key assignment - an independent label is assigned to each node in the binary tree. However, these labels are used quite differently - in the multicast re-keying LKH scheme some of these labels change at every revocation. In the Complete Subtree method labels are *static*; what changes is a single session key.

Consider an extension of the LKH scheme which we call the *Clumped re-keying method*: here, r revocations are performed at a time. For a batch of r revocations, no label is changed more than once, i.e. only the "latest" value is transmitted and used. In this variant the number of encryptions is roughly the same as in the Complete Subtree method, but it requires $\log N$ decryptions at the user, (as opposed to a single decryption in our framework). An additional advantage of the Complete Subtree method is the separation of the labels and the session key which has a consequence on the message length; see discussion at Section 4.1.

[4]This is relevant when the data is on a disk, rather then being broadcast, since broadcast results in scanning the list anyhow

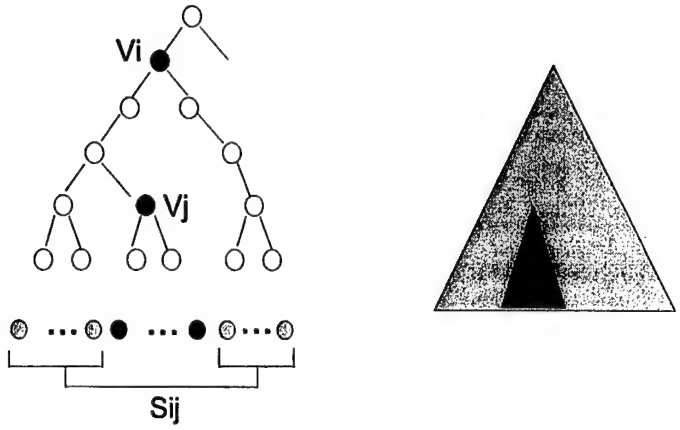


Figure 2: The Subset Difference Method: Subset $S_{i,j}$ contains all marked leaves (non-black).

3.2 The Subset Difference Method

The main disadvantage of the Complete Subtree method is that $\mathcal{N} \setminus \mathcal{R}$ may be partitioned into a number of subsets that is too large. The goal is now to reduce the partition size. We show an improved method that partitions the non-revoked receivers into at most $2r - 1$ subsets (or $1.25r$ on average), thus getting rid of a $\log N$ factor and effectively reducing the message length accordingly. In return, the number of keys stored by each receiver increases by a factor of $\frac{1}{2} \cdot \log N$. The key characteristic of the Subset-Difference method, which essentially leads to the reduction in message length, is that in this method any user belongs to *substantially* more subsets than in the first method ($O(N)$ instead of $\log N$). The challenge is then to devise an efficient procedure to succinctly encode this large set of keys at the user.

The subset description

As in the previous method, the receivers are viewed as leaves in a complete binary tree. The collection of subsets $S_1, \ldots, S_w$ defined by this algorithm corresponds to subsets of the form "a group of receivers G_1 minus another group G_2", where $G_2 \subset G_1$. The two groups G_1, G_2 correspond to leaves in two full binary subtrees. Therefore a valid subset S is represented by two nodes in the tree (v_i, v_j) such that v_i is an ancestor of v_j. We denote such subset as $S_{i,j}$. A leaf u is in $S_{i,j}$ iff it is in the subtree rooted at v_i but *not* in the subtree rooted at v_j, or in other words $u \in S_{i,j}$ iff v_i is an ancestor of u but v_j is not. Figure 2 depicts $S_{i,j}$. Note that all subsets from the Complete Subtree Method are also subsets of the Subset Difference Method; specifically, a subtree appears here as the difference between its parent and its sibling. The only exception is the full tree itself, and we will add a special subset for that. We postpone the description of the key assignment till later; for the time being assume that each subset $S_{i,j}$ has an associated key $L_{i,j}$.

The Cover

For a given set $\mathcal{R}$ of revoked receivers, let $u_1, \ldots, u_r$ be the leaves corresponding to the elements in $\mathcal{R}$. The Cover is a collection of disjoint subsets $S_{i_1,j_1}, S_{i_2,j_2} \ldots, S_{i_m,j_m}$ which partitions $\mathcal{N} \setminus \mathcal{R}$. Below is an

algorithm for finding the cover, and an analysis of its size (number of subsets).

Finding the Cover: The method partitions $\mathcal{N} \setminus \mathcal{R}$ into disjoint subsets $S_{i_1,j_1}, S_{i_2,j_2} \ldots, S_{i_m,j_m}$ as follows: let $ST(\mathcal{R})$ be the (directed) Steiner Tree induced by $\mathcal{R}$ and the root. We build the subsets collection iteratively, maintaining a tree T which is a subtree of $ST(\mathcal{R})$ with the property that any $u \in \mathcal{N} \setminus \mathcal{R}$ that is below a leaf of T has been covered. We start by making T be equal to $ST(\mathcal{R})$ and then iteratively remove nodes from T (while adding subsets to the collection) until T consists of just a single node:

1. Find two leaves v_i and v_j in T such that the least-common-ancestor v of v_i and v_j does not contain any other leaf of T in its subtree. Let v_l and v_k be the two children of v such that v_i a descendant of v_l and v_j a descendant of v_k. (If there is only one leaf left, make $v_i = v_j$ to the leaf, v to be the root of T and $v_l = v_k = v$.)
2. If $v_l \not\equiv v_i$ then add the subset $S_{l,i}$ to the collection; likewise, if $v_k \not\equiv v_j$ add the subset $S_{k,j}$ to the collection.
3. Remove from T all the descendants of v and make it a leaf.

An alternative description of the cover algorithm is as follows: consider maximal chains of nodes with outdegree 1 in $ST(\mathcal{R})$. More precisely, each such chain is of the form $[v_{i_1}, v_{i_2}, \ldots v_{i_\ell}]$ where (i) all of $v_{i_1}, v_{i_2}, \ldots v_{i_{\ell-1}}$ have outdegree 1 in $ST(\mathcal{R})$ (ii) v_{i_ℓ} is either a leaf or a node with outdegree 2 and (iii) the parent of v_{i_1} is either a node of outdegree 2 or the root. For each such chain where $\ell \geq 2$ add a subsets S_{i_1,i_ℓ} to the cover. Note that all nodes of outdegree 1 in $ST(\mathcal{R})$ are members of precisely one such chain.

The cover size: Lemma 3 shows that a cover can contain at most $2r-1$ subsets for any set of r revocations. Furthermore, if the set of revoked leaves is *random*, then the average number of subsets in a cover is $1.25r$.

Lemma 3 *Given any set of revoked leaves $\mathcal{R}$, the above method partitions $\mathcal{N} \setminus \mathcal{R}$ into at most $2r-1$ disjoint subsets.*

Proof: Every iteration increases the number of subsets by at most two (in step 2) and reduces the number of the Steiner leaves by one (in Step 3), except the last iteration that may not reduce the number of leaves but adds only one subset. Starting with r leaves, the process generates the total of $2r-1$ subsets. Moreover, every non-revoked u is in exactly one subset, the one defined by the first chain of nodes of outdegree 1 in $ST(R)$ that is encountered while moving from u towards the root. This encounter must hit a non-empty chain, since the path from u to the root cannot join $ST(R)$ in an outdegree 2 node, since this implies that $u \in \mathcal{R}$. □

The next lemma is concerned with covering more general sets than those obtained by removing users. Rather it assumes that we are removing a collection of subsets from the Subset Difference collection. It is applied later in Sections 4.2 and 5.2.

Lemma 4 *Let $\mathcal{S} = S_{i_1}, S_{i_2}, \ldots S_{i_m}$ be a collection of m* **disjoint** *subsets from the underlying collection defined by the Subset Difference method, and $\mathcal{U} = \cup_{j=1}^{m} S_{i_j}$. Then the leaves in $\mathcal{N} \setminus \mathcal{U}$ can be covered by at most $3m-1$ subsets from the underlying Subset Difference collection.*

Proof: The proof is by induction on m. When $m = 1$, $\mathcal{S}$ contains a single set. Let this set be $S_{a,b}$, which is the set that is represented by two nodes in the tree (v_a, v_b). Denote by v_c and $v_{c'}$ the parent and the sibling of v_b respectively (it is possible that $v_a \equiv v_c$), and by r the root of the tree. Then the leaves in $\mathcal{N} \setminus \mathcal{U}$ are covered by the following two sets $S_{r,a}$ and $S_{c,c'}$. If $v_a \equiv v_c$ then the cover consists of a single set, $S_{r,c'}$.

To handle the case where $m > 1$, we need the following definition. We say that a set $S_{x,y}$ is *nested* within the set $S_{a,b}$ if the tree node v_x is contained in the subtree rooted at v_b. Note that if two subsets $S_{a,b}$

and $S_{a',b'}$ are disjoint but not nested then the subtrees rooted at v_a and $v_{a'}$ must be disjoint[5]. Consider the following two cases:

1. All sets in $\mathcal{S}$ are maximal with respect to the nesting property. Let $S_{i_j} = S_{a_j,b_j}$ be the j^{th} set in $\mathcal{S}$. A cover for $\mathcal{N} \setminus \mathcal{U}$ is constructed by first covering all the subtrees rooted at the v_{b_j}'s, and then covering the rest of the leaves that are not contained in any one of the subtrees rooted at v_{a_j}. That is, for each set S_{a_j,b_j} in $\mathcal{S}$, construct the set $S_{c,c'}$ where v_c and $v_{c'}$ are the parent and the sibling of v_{b_j} respectively for the total of m sets. To cover the rest, treat the nodes $v_{a_1}, \ldots, v_{a_m}$ as m revoked leaves and apply Lemma 3 to cover this tree. This requires $2m - 1$ additional sets, hence the number of sets required to cover $\mathcal{N} \setminus \mathcal{U}$ in this case is $3m - 1$.

2. $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ such that $|\mathcal{S}_1| = k \geq 1$ and there exists a maximal set $S_{a,b} \in \mathcal{S}_2$ with respect to the nesting property such that all sets in $\mathcal{S}_1$ are nested within $S_{a,b}$. Let $\mathcal{U}'$ be the subtree rooted at v_b. The idea is to first cover $\mathcal{N} \setminus (\mathcal{U} \cup \mathcal{U}')$ and then cover the leaves in $\mathcal{U}' \setminus \mathcal{U}$. The first part of the cover can be obtained by applying the lemma recursively with the input $\mathcal{S}_2$, and the second part by applying it recursively with $\mathcal{S}_1$. By the induction hypothesis, this requires the total number of $3(m-k) - 1 + 3k - 1 = 3m - 2$ sets.

□

Average-case analysis: The analysis of Lemma 3 is a worst-case analysis and there are instances which achieve actually require $2r - 1$ sets. However, it is a bit pessimistic in the sense that it ignores the fact that a chain of nodes of outdegree 1 in $ST(\mathcal{R})$ may consist only of the end point, in which case no subset is generated. This corresponds to the case where $v_l \equiv v_i$ or $v_r \equiv v_j$ in Step 2. Suppose that the revoked set $\mathcal{R}$ is selected at random from all subsets of cardinality r of $\mathcal{N}$, then what is the expected number of subsets generated? The question is how many outdegree 1 chains are empty (i.e. contain only one point). We can bound it *from above* as follows: consider any chain for which it is known that there are k members beneath it. Then the probability that the chain is *not* empty is at most $2^{-(k-1)}$. For any $1 \leq k \leq r$ there can be at most r/k chains such that there are k leaves beneath it, since no such chain can be ancestor of another chain with k descendants. Therefore the expected number of non-empty chains is bounded by

$$\sum_{k=1}^{r} \frac{r}{k} \cdot \frac{1}{2^{k-1}} \leq 2r \sum_{k=1}^{\infty} \frac{1}{k} \cdot \frac{1}{2^k} \leq 2 \ln 2 \cdot r \approx 1.38 \cdot r.$$

Simulation experiments have shown a tighter bound of $1.25r$ for the random case. So the actual number of subsets used by the Subset Difference scheme is expected to be slightly lower than the $2r - 1$ worst case result.

Key assignment to the subsets

We now define what information each receiver must store. If we try and repeat the information-theoretic approach of the previous scheme where each receiver needs to store *explicitly* the keys of all the subsets it belongs to, the storage requirements would expand tremendously: consider a receiver u; for each complete subtree T_k it belongs to, u must store a number of keys proportional to the number of nodes in the subtree T_k that are *not* on the path from the root of T_k to u. There are $\log N$ such trees, one for each height $1 \leq k \leq \log N$, yielding a total of $\sum_{k=1}^{\log N} (2^k - k)$ which is $O(N)$ keys.

[5]The only exception is the case where b and b' are siblings and are both children of a. This is a degenerate case, and the two subsets should be replaced by a new subset $S_{a,a'}$

We therefore devise a key assignment method that requires a receiver to store only $O(\log N)$ keys per subtree, for the total of $O(\log^2 N)$ keys.

While the total number of subsets to which a user u belongs is $O(N)$, these can be grouped into $\log N$ clusters defined by the first subset i (from which another subsets is subtracted). The way we proceed with the keys assignment is to choose for each $1 \leq i \leq N$ corresponding to an internal node in the full binary tree a random and independent value LABEL_i. This value should *induce* the keys for all legitimate subsets of the form $S_{i,j}$. The idea is to employ the method used by Goldreich, Goldwasser and Micali [28] to construct pseudo-random functions, which was also used by Fiat and Naor [23] for purposes similar to ours.

Let G be a (cryptographic) pseudo-random sequence generator (see definition below) that *triples* the input, i.e. whose output length is *three times* the length of the input; let $G_L(S)$ denote the left third of the output of G on seed S, $G_R(S)$ the right third and $G_M(S)$ the middle third. We say that $G : \{0,1\}^n \mapsto \{0,1\}^{3n}$ is a pseudo-random sequence generator if no polynomial-time adversary can distinguish the output of G on a randomly chosen seed from a truly random string of similar length. Let ε_4 denote the bound on the distinguishing probability.

Consider now the subtree T_i (rooted at v_i). We will use the following top-down labeling process: the root is assigned a label LABEL_i. Given that a parent was labeled S, its two children are labeled $G_L(S)$ and $G_R(S)$ respectively. Let $\text{LABEL}_{i,j}$ be the label of node v_j derived in the subtree T_i from LABEL_i. Following such a labeling, the key $L_{i,j}$ assigned to set $S_{i,j}$ is G_M of $\text{LABEL}_{i,j}$. Note that each label induces three parts: G_L - the label for the left child, G_R - the label for the right child, and G_M the key at the node. The process of generating labels and keys for a particular subtree is depicted in Figure 3. For such a labeling process, given the label of a node it is possible to compute the labels (and keys) of all its descendants. On the other hand, without receiving the label of an ancestor of a node, its label is pseudo-random and for a node j, given the labels of all its descendants (but not including itself) the key $L_{i,j}$ is pseudo-random ($\text{LABEL}_{i,j}$, the label of v_j, is not pseudo-random given this information simply because one can check for consistency of the labels). It is important to note that given LABEL_i, computing $L_{i,j}$ requires at most $\log N$ invocations of G.

We now describe the information I_u that each receiver u gets in order to derive the key assignment described above. For each subtree T_i such that u is a leaf of T_i the receiver u should be able to compute $L_{i,j}$ iff j is *not* an ancestor of u. Consider the path from v_i to u and let $v_{i_1}, v_{i_2}, \ldots v_{i_k}$ be the nodes just "hanging off" the path, i.e. they are adjacent to the path but not ancestors of u (see Figure 3). Each j in T_i that is not an ancestor of u is a descendant of one of these nodes. Therefore if u receives the labels of $v_{i_1}, v_{i_2}, \ldots v_{i_k}$ as part of I_u, then invoking G at most $\log N$ times suffices to compute $L_{i,j}$ for any j that is not an ancestor of u.

As for the total number of keys (in fact, labels) stored by receiver u, each tree T_i of depth k that contains u contributes $k-1$ keys (plus one key for the case where there are no revocations), so the total is

$$1 + \sum_{k=1}^{\log N+1} k - 1 = 1 + \frac{(\log N + 1)\log N}{2} = \frac{1}{2}\log^2 N + \frac{1}{2}\log N + 1$$

Decryption Step: At decryption time, a receiver u first finds the subset $S_{i,j}$ such that $u \in S_{i,j}$, and computes the key corresponding to $L_{i,j}$. Using the techniques described in the complete subtree method for table lookup structure, this subset can be found in $O(\log \log N)$. The evaluation of the subset key takes now at most $\log N$ applications of a pseudo-random generator. After that, a single decryption is needed.

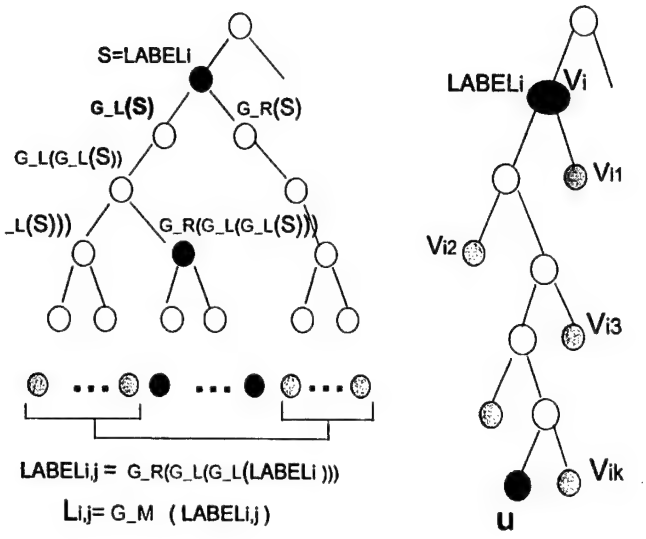


Figure 3: Key Assignment in the Subset Difference Method. *Left:* generation of $\mathrm{LABEL}_{i,j}$ and the key $L_{i,j}$. *Right:* leaf u receives the labels of $v_{i_1}, \ldots v_{i_k}$ that are induced by the label LABEL_i of v_i.

Security

In order to prove security we show that the key-indistinguishability condition (Definition 9 of Section 6) holds for this method, namely that each key is indistinguishable from a random key for all users not in the corresponding subset. Theorem 12 of Section 6 proves that this condition implies the security of the algorithm.

Observe first that for any $u \in \mathcal{N}$, u never receives keys that correspond to subtrees to which it does not belong. Let S_i denote the set of leaves in the subtree T_i rooted at v_i. For any set $S_{i,j}$ the key $L_{i,j}$ is (information theoretically) independent of all I_u for $u \notin S_i$. Therefore we have to consider only the combined secret information of all $u \in S_j$. This is specified by at most $\log N$ labels - those hanging on the path from v_i to v_j plus the two children of v_j - which are sufficient to derive all other labels in the combined secret information. Note that these labels are $\log N$ strings that were generated independently by G, namely it is never the case that one string is derived from another. Hence, a hybrid argument implies that the probability of distinguishing $L_{i,j}$ from random can be at most $\varepsilon_4 / \log N$, where ε_4 is the bound on distinguishing outputs of G from random strings.

Theorem 5 *The Subset Difference method requires (i) message length of at most* $2r - 1$ *keys (ii) to store* $\frac{1}{2}\log^2 N + \frac{1}{2}\log N + 1$ *keys at a receiver and (iii)* $O(\log N)$ *operations plus a* single *decryption operation to decrypt a message. Moreover, the method is secure in the sense of Definition 11.*

3.3 Lower Bounds

Generic lower bound

Any ciphertext in a revocation system when r users are revoked should clearly encode the original message plus the revoked subset, since it is possible to test which users decrypt correctly and which incorrectly using the preassigned secret information only (that was chosen independently of the transmitted message). Therefore we have a "generic" lower bound of $\log \binom{N}{r} \approx r \log N$ bits on the length of the header (or extra bits). Note that the subset difference method approaches this bound - the number of extra bits there is $O(r \cdot$ key-size$)$.

Lower bounds for the information-theoretic case

If the keys to all the subsets are chosen independently (and hence u explicitly receives in I_u all L_i such that $u \in S_i$) then Luby and Staddon's lower bound for the "Or Protocol" [31] can be applied. They used the Sunflower Lemma (see below) to show that any scheme which employs m subsets to revoke r users must have at least one member with at least $\frac{\binom{N}{r}^{1/m}}{mr}$ keys. This means that if we want the number of subsets m to be at most r, then the receivers should store at least $\Omega(N/r^3)$ keys (as $\binom{N}{r} \geq \left(\frac{N}{r}\right)^r$). In the case where $r \ll N$, our (non-information-theoretic) Subset Difference method does better than this lower bound.

Note that when the number of subsets used in a broadcast is $O(r \log N)$ (as it is in the Complete Subtree method) then the above bound becomes useless. We now show that even if one is willing to use this many subsets (or even more), then at least $\Omega(\log N)$ keys should be stored by the receivers. We recall the Sunflower Lemma of Erdos and Rado (see [21]).

Definition 6 *Let $S_1, S_2, \ldots, S_\ell$ be subsets of some underlying finite ground set. We say that they are a* sunflower *if the intersections of any pair of the subsets are equal, in other words, for all $1 \leq i < j \leq \ell$ we have $S_i \cap S_j = \bigcap_{i=1}^{\ell} S_i$.*

The Sunflower Lemma says that in every set system there exists a sufficiently large sunflower: in a collection of N subsets each of size at most k there exists a sunflower consisting of at least $\frac{N^{1/k}}{k}$ subsets.

Consider now the sets $T_1, T_2, \ldots T_N$ of keys the receivers store. I.e. $T_u = \{L_i | u \in S_i\}$. If for all u we have that $|T_u| \leq k$, then there exists a sunflower of $\frac{N^{1/k}}{k}$ subsets. Pick one u such that T_u is in the sunflower and make $\mathcal{R} = \{u\}$. This means that in order to cover the other members of the sunflower we must use at least $\frac{N^{1/k}}{k} - 1$ keys, since no S_i can be used to cover two of the other members of the sunflower (otherwise S_i must also have the revoked u as a member). This means, for instance, that if $k = \sqrt{\log N}$ then just to revoke a single user requires using at least $\frac{2^{\sqrt{\log N}}}{\sqrt{\log N}} - 1$ subsets.

4 Further Discussions

4.1 Implementation Issues

Implementing E_L and F_K

One of the issues that arises in implementing a Subset-Cover scheme is how to implement the two cryptographic primitives E_L and F_K. The basic requirements from E_L and F_K were outlined above in Section 2. However, it is sometimes desirable to chose an encryption F that might be weaker (uses shorter keys)

than the encryption chosen for E. The motivation for that is twofold: (1) to speedup the decoding process at the receiver (2) to shorten the length of the header. Such a strategy makes sense, for example, for copyright protection purposes. There it may not make sense to protect a *specific* ciphertext so that breaking it is very expensive; on the other hand we do want to protect the long lived keys of the system with a strong encryption scheme.

Suppose that F is implemented by using a stream cipher with a long key, but sending some of its bits in the clear; thus K corresponds to the hidden part of the key and this is the only part that needs to be encrypted in the header. (One reason to use F in such a mode rather than simply using a method designed with a small key is to prevent a preprocessing attack against F.) This in itself does not shorten the header, since it depends on the block-length of E (assuming E is implemented by block-cipher). We now provide a specification for using E, called Prefix-Truncation, which reduces the header length as well, in addition to achieving speedup, without sacrificing the security of the long-lived keys. Let $\text{Prefix}_i S$ denote the first i bits of a string S. Let E_L be a block cipher and $\mathcal{U}$ be a random string whose length is the length of the block of E_L. Let K be a relatively short key for the cipher F_K (whose length is, say, 56 bits). Then, $[\text{Prefix}_{|K|} E_L(\mathcal{U})] \oplus K$ provides an encryption that satisfies the definition of E as described in Section 6. The Prefix-Truncated header is therefore:

$$\langle [\, i_1, i_2, \ldots, i_m, \mathcal{U}, [\text{Prefix}_{|K|} E_{L_{i_1}}(\mathcal{U})] \oplus K, \ldots, [\text{Prefix}_{|K|} E_{L_{i_m}}(\mathcal{U})] \oplus K \,], F_K(M) \rangle$$

Note that this reduces the length of the header down to about $m \times |K|$ bits long (say $56m$) instead of $m \times |L|$. In the case where the key length of E is marginal, then the following heuristic can be used to remove the factor m advantage that the adversary has in a brute-force attack which results from encrypting the same string $\mathcal{U}$ with m different keys. Instead, encrypt the string $\mathcal{U} \oplus i_j$, namely

$$\langle [\, i_1, i_2, \ldots, i_m, \mathcal{U}, [\text{Prefix}_{|K|} E_{L_{i_1}}(\mathcal{U} \oplus i_1)] \oplus K, \ldots, [\text{Prefix}_{|K|} E_{L_{i_m}}(\mathcal{U} \oplus i_m)] \oplus K \,], F_K(M) \rangle$$

All-Or-Nothing Encryptions for F_K

As before, we can imagine cases where the key used by F_K is only marginally long enough. Moreover, in a typical scenario like copyright protection, the message M is long (e.g. M may be a title on a CD or a DVD track). In such cases, it is possible to extract more security from the long message for a fixed number of key bits using the All-Or-Nothing encryption mode originally suggested by [40]. These techniques assure that the entire ciphertext must be decrypted before even a single message block can be determined. The concrete method of [40] results in a penalty of a factor of three in the numbers encryptions/decryptions required by a legitimate user; however, for a long message that is composed of l blocks, a brute-force attack requires a factor of l more time than a similar attack would require otherwise. Other All-Or-Nothing methods can be applied as well.

The drawback of using an All-Or-Nothing mode is its *latency*, namely the entire message M must be decoded before the first block of plaintext is known. This makes the technique unusable for applications that cannot tolerate such latency.

Frequently Refreshed Session Keys

Suppose that we want to prevent an illegal redistribution channel that will use some low bandwidth means to send K, the session key (a low bandwidth label or a bootlegged CD). A natural approach to combat such channel is to encode different parts of the message M with different session keys, and to send all different

session keys encrypted with all the subset keys. That is, send $l > 1$ different session keys all encrypted with the same cover, thus increasing the length of the header by a factor of l. This means that in order to have only a modest increase in the header information it is important that m, the number of subsets, will be as small as possible. Note that the number of decryptions that the receiver needs to perform in order to obtain its key $\mathcal{L}_{i_j}$ which is used in this cover remains one.

Storage at the Center

In both the Complete Subtree and Subset Difference methods, a unique label is associated with each node in the tree. Storing these labels explicitly at the Center can become a serious constraint. However, these labels can be generated at the center by applying a pseudo-random function on the name of the node without affecting the security of the scheme. This reduces the storage required by at the Center to the *single* key of the pseudo-random function.

Furthermore, it may be desirable to distribute the center between several servers with the objective of avoiding a single or few points of attack. In such a case the distributed pseudo-random functions of [36] may be used to define the labels.

4.2 Hierarchical Revocation

Suppose that the receivers are grouped in a hierarchical manner, and that it is desirable to revoke a group that consists of the subordinates of some entity, without paying a price proportional to the group size (for instance all the players of a certain manufacturer). Both methods of Section 3 lend themselves to hierarchical revocation naturally, given the tree structure. If the hierarchy corresponds to the tree employed by the methods, then to revoke the receivers below a certain node counts as just a single user revocation.

By applying Lemma 4 we get that in the Subset Difference Method we can remove any collection of m subsets and cover the rest with $3m - 1$ subsets. Hence, the hierarchical revocation can be performed by first constructing m sets that cover all revoked devices, and then covering all the rest with $3m - 1$, yielding the total of $4m$ sets.

4.3 Public Key methods

In some scenarios it is desireable to use a revocation scheme in a public-key mode, i.e. when the party that generates the ciphertext is not necessarily trustworthy and should not have access to the decryption keys of the users, or when ciphertexts may be generated by a number of parties. Any Subset-Cover revocation algorithm can be used in this mode: the Center (a trusted entity) generates the private-keys corresponding to the subsets and hands each user the private keys it needs for decryption. The (non necessarily trusted) party that generates the ciphertext should only have access to public-keys corresponding to the subsets which we call "the public-key file". That is, E is a public key cryptosystem whereas F is as before. In principal, any public key encryption scheme with sufficient security can be used for E. However, not all yield a system with a reasonable efficiency. Below we discuss the problems involved, and show that a Diffie-Hellman type scheme best serves this mode.

Public Key Generation: Recall that the Subtree Difference method requires that subset keys are derived from labels. If used in a public-key mode, the derivation yields random bits that are then used to generate the private/public key pair. For example, if RSA keys are used, then the random strings that are generated by the Pseudo Random Generator G can be used as the random bits which are input to the procedure which generates an RSA key. However, this is rather complicated, both in terms of the bits and time needed.

Therefore, whenever the key assignment is not information-theoretic it is important to use a public-key scheme where the mapping from random bits to the keys is efficient. This is the case with the Diffie-Hellman scheme.

Size of Public Key File: The problem is that the public key file might be large, proportional to w, the number of subsets. In the Complete Subtree method $w = 2N - 1$ and in the Subtree Difference method it is $N \log N$. An interesting open problem is to come up with a public-key cryptosystem where it is possible to compress the public-keys to a more manageable size. For instance, an identity-based cryptosystem would be helpful for the information-theoretic case where keys are assigned independently. A recent proposal that fits this requirement is [8].

Prefix-Truncated Headers: We would like to use the Prefix-Truncation, described in Section 4.1, with public-key cryptosystem to reduce the header size without sacrificing security of long-term keys. It can not be employed with an arbitrary public key cryptosystem (e.g. RSA). However, a Diffie-Hellman public key system which can be used for the Prefix-Truncation technique can be devised in the following manner.

Let G be a group with a generator g and let the subset keys be $L_1 = y_1, L_2 = y_2, \ldots, L_w = y_w$ elements in G. Let $g^{y_1}, g^{y_2}, \ldots, g^{y_w}$ be their corresponding public keys. Define h as a pairwise-independent function $h : G \mapsto \{0,1\}^{|K|}$ that maps elements which are randomly distributed over G to randomly distributed strings of the desired length (see e.g. [37] for a discussion of such functions). Given the subsets $S_{i_1}, \ldots, S_{i_m}$ to be used in the header, the encryption E can be done by picking a new element x from G, publicizing g^x, and encrypting K as $E_{L_{i_j}}(K) = h(g^{xy_{i_j}}) \oplus K$. That is, the header now becomes

$$\langle [\, i_1, i_2, \ldots, i_m, g^x, h,\ h(g^{xy_{i_1}}) \oplus K, \ldots, h(g^{xy_{i_m}}) \oplus K \,], F_K(M) \rangle$$

Interestingly, in terms of the broadcast length such system hardly increases the number of bits in the header as compared with a shared-key system - the only difference is g^x and the description of h. Therefore this difference is fixed and does not grow with the number of revocations. Note however that the scheme as defined above is not immune to chosen-ciphertext attacks, but only to chosen plaintext ones. Coming up with public-key schemes where prefix-truncation is possible that are immune to chosen ciphertext attacks of either kind is an interesting challenge[6].

4.4 Applications to Multicast

The difference between key management for the scenario considered in this paper and for the Logical Key Hierarchy for multicast is that in the latter the users (i.e. receivers) may update their keys [43, 42]. This update is referred to as a re-keying event and it requires all users to be connected during this event and change their internal state (keys) accordingly. However, even in the multicast scenario it is not reasonable to assume that all the users receive all the messages and perform the required update. Therefore some mechanism that allows individual update must be in place. Taking the stateless approach gets rid of the need for such a mechanism: simply add a header to each message denoting who are the legitimate recipients by revoking those who should not receive it. In case the number of revocations is not too large this may yield a more manageable solution. This is especially relevant when there is a single source for the sending messages or when public-keys are used.

Backward secrecy: Note that revocation in itself lacks backward secrecy in the following sense: a constantly listening user that has been revoked from the system records all future transmission (which it can't decrypt anymore) and keeps all ciphertexts. At a later point it gains a valid *new* key (by re-registering)

[6]Both the scheme of Cramer and Shoup [14] and the random oracle based scheme [25] require some specific information for each recipient; a possible approach with random oracles is to add a zk proof-of-knowledge for x.

which allows decryption of all past communication. Hence, a newly acquired user-key can be used to decrypt all past session keys and ciphertexts. The way that [43, 42] propose to achieve backward secrecy is to perform re-keying when new users are added to the group (such a re-keying may be reduced to only one way chaining, known as LKH+), thus making such operations non-trivial. We point out that in the subset-cover framework and especially in the two methods we proposed it may be easier: At any given point of the system include in the set of revoked receivers all identities that have not been assigned yet. As a result, a newly assigned user-key cannot help in decrypting an earlier ciphertext. Note that this is feasible since we assume that new users are assigned keys in a consecutive order of the leaves in the tree, so unassigned keys are consecutive leaves in the complete tree and can be covered by at most $\log N$ sets (of either type, the Complete-Subtree method or the Subtree-Difference method). Hence, the unassigned leaves can be treated with the hierarchical revocation technique, resulting in *adding* at most $\log N$ revocations to the message.

4.5 Comparison to CPRM

CPRM/CPPM (Content Protection for Recordable Media and Pre-Recorded Media) is a technology announced and developed by 4 companies (known as the 4C's): IBM, Intel, Matsushita and Toshiba [18]. It defines a method for protecting content on physical media such as recordable DVD, DVD Audio, Secure Digital Memory Card and Secure CompactFlash. A licensing Entity (the Center) provides a unique set of secret device keys to be included in each device at manufacturing time. The licensing Entity also provides a Media Key Block (MKB) to be placed on each compliant media (for example, on the DVD). The MKB is essentially the Header of the ciphertext which encrypts the session key. It is assumed that this header resides on a write-once area on the media, e.g. a Pre-embossed lead-in area on the recordable DVD. When the compliant media is placed in a player/recorder device, it computes the session key from the Header (MKB) using its secret keys; the content is then encrypted/decrypted using this session key.

The algorithm employed by CPRM is essentially a Subset-Cover scheme. Consider a table with A rows and C columns. Every device (receiver) is viewed as a collection of C entries from the table, exactly one from each column, that is $u = [u_1, \ldots, u_C]$ where $u_i \in \{0, 1, \ldots, A-1\}$. The collection of subsets $S_1, \ldots, S_w$ defined by this algorithm correspond to subsets of receivers that share the same entry at a given column, namely $S_{r,i}$ contains all receivers $u = [u_1, \ldots, u_C]$ such that $u_i = r$. For every $0 \leq i \leq A-1$ and $1 \leq j \leq C$ the scheme associates a key denoted by $L_{i,j}$. The private information I_u that is provided to a device $u = [u_1, \ldots, u_C]$ consists of C keys $L_{u_1,1}, L_{u_2,2}, \ldots, L_{u_C,C}$.

For a given set $\mathcal{R}$ of revoked devices, the method partitions $\mathcal{N} \setminus \mathcal{R}$ as follows: $S_{i,j}$ is in the cover iff $S_{i,j} \bigcap \mathcal{R} = \emptyset$. While this partition guarantees that a revoked device is never covered, there is a low probability that a non-revoked device $u \notin \mathcal{R}$ will not be covered as well and therefore become non-functional[7].

The CPRM method is a Subset-Cover method with two exceptions: (1) the subsets in a cover are not necessarily disjoint and (2) the cover is not always perfect as a non-revoked device may be uncovered. Note that the CPRM method is not *r-flexible*: the probability that a non-revoked device is uncovered grows with r, hence in order to keep it small enough the number of revocations must be bounded by A.

For the sake of comparing the performance of CPRM with the two methods suggested in this paper, assume that $C = \log N$ and $A = r$. Then, the message is composed of $r \log N$ encryptions, the storage at the receiver consists of $\log N$ keys and the computation at the receiver requires a single decryption. These bounds are similar to the Complete Subtree method; however, unlike CPRM, the Complete Subtree method is r-flexible and achieves perfect coverage. The advantage of the Subset Difference Method is much more substantial: in addition to the above, the message consists of $1.25r$ encryptions on average, or of at most $2r - 1$ encryptions, rather than $r \log N$.

[7] This is similar to the scenario considered in [27]

For example, in DVD Audio, the amount of storage that is dedicated for its MKB (the header) is 3 MB. This constraints the maximum allowed message length. Under a certain choice of parameters, such as the total number of manufactured devices and the number of distinct manufacturers, with the current CPRM algorithm the system can revoke up to about 10,000 devices. In contrast, for the same set of parameters and the same 3MB constraint, a Subset-Difference algorithm achieves up to 250,00 (!) revocations, a factor of 25 improvement over the currently used method. This major improvement is partly due to fact that hierarchical revocation can be done very effectively, a property that the current CPRM algorithm does not have.

5 Tracing Traitors

It is highly desirable that a revocation mechanism could work in tandem with a tracing mechanism to yield a *trace and revoke* scheme. We show a tracing method that works for many schemes in the subset-cover framework. The method is quite efficient. The goal of a tracing algorithm is to find the identities of those that contributed their keys to an illicit decryption box and revoke them; short of identifying them we should render the box useless by finding a "pattern" that does not allow decryption using the box, but still allows broadcasting to the legitimate users. Note that this is a slight relaxation of the requirement of a tracing mechanism, say in [34] (which requires an identification of the traitor's identity) and in particular it lacks *self enforcement* [20]. However as a mechanism that works in conjunction with the revocation scheme it is a powerful tool to combat piracy.

The model

Suppose that we have found an illegal decryption-box (decoder, or clone) which contains the keys associated with at most t receivers $u_1, \ldots, u_t$ known as the "traitors".

We are interested in "black-box" tracing, i.e. one that does not take the decoder apart but by providing it with an encrypted message and observing its output (the decrypted message) tries to figure out who leaked the keys. A pirate decoder is of interest if it correctly decodes with probability p which is at least some threshold q, say $q > 0.5$. We assume that the box has a "reset button", i.e. that its internal state may be retrieved to some initial configuration. In particular this excludes a "locking" strategy on the part of the decoder which says that in case it detects that it is under test, it should refuse to decode further. Clearly software-based systems can be simulated and therefore have the reset property.

The result of a tracing algorithm is either a subset consisting of traitors or a partition into subsets that renders the box useless i.e. given an encryption with the given partition it decrypts with probability smaller than the threshold q while all good users can still decrypt.

In particular, a "subsets based" tracing algorithm devises a sequence of queries which, given a black-box that decodes with probability above the threshold q, produces the results mentioned above. It is based on constructing useful sets of revoked devices $\mathcal{R}$ which will finally allow the detection of the receiver's identity or the configuration that makes the decoder useless. A tracing algorithm is evaluated based on (i) the level of performance downgrade it imposes on the revocation scheme (ii) number of queries needed.

5.1 The Tracing Algorithm

Subset tracing: An important procedure in our tracing mechanism is one that given a partition $\mathcal{S} = S_{i_1}, S_{i_2}, \ldots S_{i_m}$ and an illegal box outputs one of two possible outputs: either (1) that the box cannot decrypt with probability greater than the threshold when the encryption is done with partition $\mathcal{S}$ or (ii) Finds

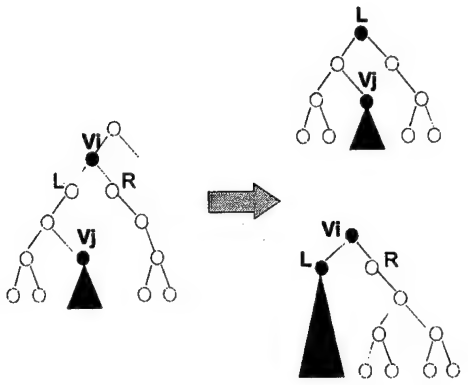


Figure 4: Bifurcating a Subset Difference set $S_{i,j}$, depicted in the left. The black triangle indicates the excluded subtree. L and R are the left and the right children of v_i. The resulting sets $S_{L,j}$ and $S_{i,L}$ are depicted to the right.

a subset S_{i_j} such that S_{i_j} contains a traitor. Such a procedure is called subset tracing. We describe it in detail in Section 5.1.1.

Bifurcation property: Given a subset-tracing procedure, we describe a tracing strategy that works for many Subset-Cover revocation schemes. The property that the revocation algorithm should satisfy is that for any subset $S_i, 1 \leq i \leq w$, it is possible to partition S_i into two (or constant) roughly equal sets, i.e. that there exists $1 \leq i_1, i_2 \leq w$ such that $S_i = S_{i_1} \cup S_{i_2}$ and $|S_{i_1}|$ is roughly the same as $|S_{i_2}|$. For a Subset Cover scheme, let the *bifurcation value* be the relative size of the largest subset in such a split.

Both the Complete Subtree and the Subtree Difference methods satisfy this requirement: in the case of the Complete Subtree Method each subset, which is a complete subtree, can be split into exactly two equal parts, corresponding to the left and right subtrees. Therefore the bifurcation value is $1/2$. As for the Subtree Difference Method, Each subset $S_{i,j}$ can be split into two subsets each containing between one third and two thirds of the elements. Here, again, this is done using the left and right subtrees of node i. See Figure 4. The only exception is when i is a parent of j, in which case the subset is the complete subtree rooted at the other child; such subsets can be perfectly split. The worst case of $(1/3, 2/3)$ occurs when i is the grandparent of j. Therefore the bifurcation value is $2/3$.

The Tracing Algorithm: We now describe the general tracing algorithm, assuming that we have a good subset tracing procedure. The algorithm maintains a partition $S_{i_1}, S_{i_2}, \ldots S_{i_m}$. At each phase one of the subsets is partitioned, and the goal is to partition a subset only if it contains a traitor.

Each phase initially applies the subset-tracing procedure with the current partition $\mathcal{S} = S_{i_1}, S_{i_2}, \ldots S_{i_m}$. If the procedure outputs that the box cannot decrypt with $\mathcal{S}$ then we are done, in the sense that we have found a way to disable the box without hurting any legitimate user. Otherwise, let S_{i_j} be the set output by the procedure, namely S_{i_j} contains the a traitor.

If S_{i_j} contains only one possible candidate - it must be a traitor and we permanently revoke this user; this doesn't hurt a legitimate user. Otherwise we split S_{i_j} into two roughly equal subsets and continue with the new partitioning. The existence of such a split is assured by the bifurcation property.

Analysis: Since a partition can occur only in a subset that has a traitor and contains more than one element, it follows that the number of iterations can be at most $t \log_a N$, where a is the inverse of the bifurcation

value (a more refined expression is $t(\log_a N - log_2 t)$, the number of edges in a binary tree with t leaves and depth $\log_a N$.)

5.1.1 The Subset Tracing Procedure

The Subset Tracing procedure first tests whether the box decodes with sufficiently high probability (say greater than 0.5) when the partition is $\mathcal{S} = S_{i_1}, S_{i_2}, \ldots S_{i_m}$. If not, then it concludes (and outputs) that the box cannot decrypt with $\mathcal{S}$. Otherwise, it needs to find a subset S_{i_j} that contains a traitor.

Let p_j be the probability that the box decodes the ciphertext

$$\langle [i_1, i_2, \ldots, i_m, E_{L_{i_1}}(R_K), E_{L_{i_2}}(R_K), \ldots, E_{L_{i_j}}(R_K), E_{L_{i_{j+1}}}(K), \ldots, E_{L_{i_m}}(K)], F_K(M) \rangle$$

where R_K is a random string of the same length as the key K. That is, p_j is the probability of decoding when the first j subsets are noisy and the remaining subsets encrypt the correct key. Note that $p_0 = p$ and $p_m = 0$, hence there must be some $0 < j \leq m$ for which $|p_{j-1} - p_j| \geq \frac{p}{m}$.

Claim 7 *Let ϵ be an upper bound on the sum of the probabilities of breaking the encryption scheme E and key assignment method. If p_{j-1} is different from p_j by more than ϵ, then the set S_{i_j} must contain a traitor.*

Proof: From the box's point of view, a ciphertext that contains $j-1$ noisy subsets is different from a ciphertext that contains j noisy subsets only if the box is able to distinguish between $E_{L_{i_j}}(K)$ and $E_{L_{i_j}}(R_K)$. Since this cannot be due to breaking the encryption scheme or the key assignment method alone, it follows that the box must contain L_{i_j}. □

We now describe a binary-search-like method that efficiently finds a pair of values p_j, p_{j-1} among $p_0, \ldots, p_m$ satisfying $|p_{j-1} - p_j| \geq \frac{p}{m}$. Starting with the entire interval $[1, m]$, the search is repeatedly narrowed down to an arbitrary interval $[a, b]$. At each stage, the middle value $p_{\frac{a+b}{2}}$ is computed (approximately) and the interval is further halved either to the left half or to the right half, depending on difference between $p_{\frac{a+b}{2}}$ and the endpoint values p_a and p_b of the interval and favoring the interval with the larger difference. The method is outlined below; it outputs the index j.

SubsetTracing(a, b, p_a, p_b)

```
If (a == b − 1)
  return b
Else
  c = ⌈(a+b)/2⌉
  Find p_c
  If |p_c − p_a| ≥ |p_b − p_a|
    SubsetTracing(a, c, p_a, p_c)
  Else
    SubsetTracing(c, b, p_c, p_b)
```

Efficiency: Let the probability of error be ϵ and the range error be δ. Subset tracing is comprised of $\log m$ steps. At each step it should decide with probability at least $1 - \epsilon$ the following:

- If $\frac{|p_c - p_a|}{|p_b - p_a|} > \frac{1}{2}(1 + \delta)$, decide "$|p_c - p_a| > |p_b - p_c|$"

- If $\frac{|p_c-p_a|}{|p_b-p_a|} < \frac{1}{2}(1-\delta)$, decide "$|p_c-p_a| < |p_b-p_c|$"
- Otherwise, any decision is acceptable.

In order to distinguish these two cases apply Claim 8 below. Since the Claim is applied $\log m$ times, choose $\delta = \frac{1}{\log m}$. At each step with probability at least $1-\varepsilon$ the interval $|p_b - p_a|$ shrinks by at least a factor of $\frac{1}{2}(1-\delta)$, so at the i^{th} step the interval length is (with probability at least $i \cdot \epsilon$) larger than $(\frac{1}{2}(1-\delta))^i$; hence the smallest possible interval when $i = \delta = \frac{1}{\log m}$ is of length $\Delta \geq \frac{1}{em}$, with probability at least $\varepsilon \log m$. It follows that a subset tracing procedure that works with success probability of $(1-\varepsilon \log m)$ requires at most $O(m^2 \log \frac{1}{\varepsilon} \log^3 m)$ ciphertext queries to the decoding box over the entire procedure. Note that a total probability of success bounded away from zero is acceptable, since it is possible to verify that the resulting p_{j-1}, p_j differ, and hence ϵ can be $O(1/\log m)$.

Claim 8 *Let p_a, p_b be the two probabilities at the end-points of an interval $[a,b]$ such that $|p_a - p_b| \geq \Delta$, and let X_c be a random variable such that $Prob[X_c = 1] = p_c$ where p_c is unknown. We would like to sample the decoding box and decide "$|p_c-p_a| > |p_b-p_c|$" or "$|p_c-p_a| < |p_b-p_c|$" according to the definition given above. The number of samples (i.e. ciphertext queries) required to reach this conclusion with error at most ε is $O(\log(\frac{1}{\varepsilon})/(\Delta^2\delta^2))$.*

Proof: Let X_a, X_b and X_c be $\{0,1\}$ variables satisfying $P[X_a = 1] = p_a$, $P[X_b = 1] = p_b$ and $P[X_c = 1] = p_c$. The variant of Chernoff bounds described in [1], Corollary A.7 [p. 236], states that for a sequence of mutually independent random variables $Y_1, \ldots, Y_n$ satisfying $P[Y_i = 1-p] = p$ and $P[Y_i = -p] = 1-p$ it holds that $P[|\sum_i^n Y_i| > t] < 2e^{-2t^2/n}$ for $t > 0$. Suppose we want to estimate p_a by sampling n_a times from the distribution of X_a. Let p'_a be estimation that results from this sampling. Applying the above Chernoff's bound we can conclude that $P[|p'_a - p_a| > t] < 2e^{-2n(t+p_a)^2}$. Hence, by choosing $n_a = \frac{\ln \frac{2}{\varepsilon}}{2(t+p_a)^2}$, the estimated p'_a obtained from sampling n_a times satisfies $P[|p'_a - p_a| > t] < \varepsilon$. Clearly, by sampling $n = \frac{\ln \frac{2}{\varepsilon}}{2t^2} > n_a$ times the ϵ-bounded error is also achieved. Analogously, this analysis holds for the process of sampling from X_b and X_c, where p'_b and p'_c are the estimations that result from sampling the distributions X_b and X_c.

In order to decide whether "$|p_c-p_a| > |p_b-p_c|$" or "$|p_c-p_a| < |p_b-p_c|$":

- Sample $n = \frac{\ln \frac{2}{\varepsilon}}{2(\frac{\Delta\delta}{4})^2}$ times from each of the distributions X_a, X_b and X_c and compute p'_a, p'_b, p'_c, the estimations for p_a, p_b, p_c respectively.
- If $p'_c > \frac{p'_a+p'_b}{2}$ then decide "$|p_c-p_a| > |p_b-p_c|$"
- If $p'_c < \frac{p'_a+p'_b}{2}$ then decide "$|p_c-p_a| < |p_b-p_c|$"

The number of samples conducted by this procedure is $3n = O(\log(\frac{1}{\varepsilon})/(\Delta^2\delta^2))$. We now have to show that this decision is in accordance with the definition above. Note that the Chernoff bound implies that with probability $1-\epsilon$ we have (i) $p'_a \in (p_a - \frac{\Delta\delta}{4}, p_a + \frac{\Delta\delta}{4})$, (ii) $p'_b \in (p_b - \frac{\Delta\delta}{4}, p_b + \frac{\Delta\delta}{4})$ and (iii) $p'_c \in (p_c - \frac{\Delta\delta}{4}, p_c + \frac{\Delta\delta}{4})$.

If $\frac{|p_c-p_a|}{|p_b-p_a|} > \frac{1}{2}(1+\delta)$ then by substituting $\Delta \leq p_b - p_a$ we get that $p_c > \frac{p_b+p_a}{2} + \frac{\Delta\delta}{2}$. Combining this with the above, $p'_c \geq p_c - \frac{\Delta\delta}{4} > \frac{p_b+p_a}{2} + \frac{\Delta\delta}{4} \geq \frac{p'_a+p'_b}{2}$ so the correct decision is reached. Similarly, if $\frac{|p_c-p_a|}{|p_b-p_a|} < \frac{1}{2}(1-\delta)$. □

Noisy binary search: A more sophisticated procedure is to treat the Subset-Tracing procedure as *noisy binary search*, as in [22]. They showed that in a model where each answer is correct with some fixed probability (say greater than $2/3$) that is independent of history it is possible to perform a binary search in $O(\log N)$ queries. Each step might require backtracking; in the subset-tracing scenario, the procedure backtracks if the condition $|p_a - p_b| \geq (\frac{1}{2})^i$ does not hold at the i^{th} step (which indicates an error in an earlier decision). Estimating the probability values within an accuracy of $\frac{1}{m}$ while guaranteeing a constant probability of error requires only $O(m^2)$ ciphertexts queries. This effectively means that we can fix δ and ϵ to be constants (independent of m). Therefore, we can perform the noisy binary search procedure with $O(m^2 \log m)$ queries.

5.2 Improving the Tracing Algorithm

The basic traitors tracing algorithm described above requires $t \log(N/t)$ iterations. Furthermore, since at each iteration the number of subsets in the partition increases by one, tracing t traitors may result with up to $t \log(N/t)$ subsets and hence in messages of length $t \log(N/t)$. This bound holds for any Subset-Cover method satisfying the *Bifurcation property*, and both the Complete Subtree and the Subset Difference methods satisfy this property. What is the bound on the number of traitors that the algorithm can trace?

Recall that the Complete Subtree method requires a message length of $r \log(N/r)$ for r revocations, hence the tracing algorithm can trace up to r traitors if it uses the Complete Subtree method. However, since the message length of the Subset Difference method is at most $2r - 1$, only $\frac{2r-1}{\log N/r}$ traitors can be traced if Subset Difference is used. We now describe an improvement on the basic tracing algorithm that reduces the number of subsets in the partition to $5t - 1$ for the Subset Difference method (although the number of iterations remains $t \log(N/t)$). With this improvement the algorithm can trace up to $r/5$ traitors.

Note that among the $t \log N/t$ subsets generated by the basic tracing algorithm, only t actually contain a traitor. The idea is to repeatedly merge those subsets which are not known to contain a traitor.[8] Specifically, we maintain at each iteration a *frontier* of at most $2t$ subsets plus $3t - 1$ additional subsets. In the following iteration a subset that contains a traitor is further partitioned; as a result, a new *frontier* is defined and the remaining subsets are re-grouped.

Frontier subsets: Let $S_{i_1}, S_{i_2}, \ldots S_{i_m}$ be the partition at the current iteration. A pair of subsets $(S_{i_{j1}}, S_{i_{j2}})$ is said to be in the frontier if $S_{i_{j1}}$ and $S_{i_{j2}}$ resulted from a split-up of a single subset at an earlier iteration. Also neither ($S_{i_{j1}}$ nor $S_{i_{j2}}$) was singled out by the subset tracing procedure so far. This definition implies that the frontier is composed of k disjoint pairs of *buddy subsets*. Since buddy-subsets are disjoint and since each pair originated from a single subset that contained a traitor (and therefore has been split) $k \leq t$.

We can now describe the improved tracing algorithm which proceeds in iterations. Every iteration starts with a partition $\mathcal{S} = S_{i_1}, S_{i_2}, \ldots S_{i_m}$. Denote by $F \subset S$ the frontier of S. An iteration consists of the following steps, by the end of which a new partition $\mathcal{S}'$ and a new frontier F' is defined.

- As before, use the Subset Tracing procedure to find a subset S_{i_j} that contains a traitor. If the tracing procedure outputs that the box can not decrypt with $\mathcal{S}$ then we are done. Otherwise, split S_{i_j} into $S_{i_{j1}}$ and $S_{i_{j2}}$.
- $F' = F \cup S_{i_{j1}} \cup S_{i_{j2}}$ ($S_{i_{j1}}$ and $S_{i_{j2}}$ are now in the frontier). Furthermore, if S_{i_j} was in the frontier F and S_{i_k} was its buddy-subset in F then $F' = F' \setminus S_{i_k}$ (remove S_{i_k} from the frontier).

[8] This idea is similar to the second scheme of [24], Section 3.3. However, in [24] the merge is straightforward as their model allows any subset. In our model only members from the Subset Difference are allowed, hence a merge which produces subsets of this particular type is non-trivial.

- Compute a cover C for all receivers that are not covered by F'. Define the new partition S' as the union of C and F'.

To see that the process described above converges, observe that at each iteration the number of new *small* frontier sets always increases by at least one. More precisely, at the end of each iteration construct a vector of length N describing how many sets of size i, $1 \leq i \leq N$, constitute the frontier. It is easy to see that these vectors are lexicographically increasing. The process must stop when or before all sets in the frontier are singletons.

By definition, the number of subsets in a frontier can be at most $2t$. Furthermore, they are paired into at most t disjoint buddy subsets. As for non-frontier subsets (C), Lemma 4 shows that covering the remaining elements can be done by at most $|F| \leq 3t - 1$ subsets (note that we apply the lemma so as to cover all elements that are not covered by the buddy subsets, and there are at most t of them). Hence the partition at each iteration is composed of at most $5t - 1$ subsets.

5.3 Tracing Traitors from Many Boxes

As new illegal decoding boxes, decoding clones and hacked keys are continuously being introduced during the lifetime of the system, a revocation strategy needs to be adopted in response. This revocation strategy is computed by first revoking the identities (leaves) of all the receivers that need to be excluded, resulting in some partition S_0. Furthermore, to trace traitors from possibly more than one black box and make all of these boxes non-decoding, the tracing algorithm needs to be run *in parallel* on all boxes by providing all boxes with the same input. The initial input is the partition S_0 that results from direct revocation of all known identities. As the algorithm proceeds, when the *first* box detects a traitor in one of the sets it re-partitions accordingly and the new partition is now input to *all* boxes simultaneously. The output of this simultaneous algorithm is a partition (or "revocation strategy") that renders all revoked receivers and illegal black boxes invalid.

6 Security of the Framework

In this section we discuss the security of a Subset-Cover algorithm. Intuitively, we identify a critical property that is required from the key-assignment method in order to provide a secure Subset-Cover algorithm. We say that a subset-cover algorithm satisfies the "key-indistinguishability" property if for every subset S_i its key L_i is indistinguishable from a random key given all the information of all users that are *not* in S_i. We then proceed to show that any subset-cover algorithm that satisfies the key-indistinguishability property provides a secure encryption of the message.

We must specify what is a secure revocation scheme, i.e. describe the adversary's power and what is considered a successful break. We provide a sufficient condition for a Subset-Cover revocation scheme $\mathcal{A}$ to be secure. We start by stating the assumptions on the security of the encryption schemes E and F. All security definitions given below refer to an adversary whose challenge is of the form: distinguish between two cases 'i' and 'ii'.

6.1 Assumptions on the Primitives

Recall that the scheme employs two cryptographic primitives F_K and E_L. The security requirements of these two methods are different, since F_K uses short lived keys whereas E_L uses long-lived ones. In both cases we phrase the requirements in terms of a the probability of success in distinguishing an encryption

of the true message from an encryption of a random message. It is well known that such formulation is equivalent to semantic security (that anything that can be computed about the message given the ciphertext is computable without it), see [29, 28, 4][9].

The method F_K for encrypting the body of the message should obey the following property: consider any feasible adversary $\mathcal{B}$ that chooses a message M and receives for a randomly chosen $K \in \{0,1\}^\ell$ one of the following (i) $F_K(M)$ (ii) $F_K(R_M)$ for a random message R_M of length $|M|$. The probability that $\mathcal{B}$ distinguishes the two cases is negligible and we denote the bound by ε_1, i.e.

$$|\Pr[\mathcal{B} \text{ outputs 'i'} \,|F_K(M)] - \Pr[\mathcal{B} \text{ outputs 'i'} \,|F_K(R_M)]| \leq \varepsilon_1.$$

Note that implementing F_K by a pseudo-random generator (stream-cipher) where K acts as the seed and whose output is Xored bit-by bit with the message satisfies this security requirement.

The long term encryption method E_L should withstand a more severe attack, in the following sense: consider any feasible adversary $\mathcal{B}$ that for a random key L gets to adaptively choose polynomially many inputs and examine E_L's encryption and similarly provide ciphertexts and examine E_L's decryption. Then $\mathcal{B}$ is faced with the following challenge: for a random plaintext x (which is provided in the clear) it receives one of (i) $E_L(x)$ or (ii) $E_L(R_x)$ where R_x is a random string of length $|x|$. The probability that $\mathcal{B}$ distinguishes the two cases is negligible and we denote the bound by ε_2, i.e.

$$|\Pr[\mathcal{B} \text{ outputs 'i'} \,|E_L(x)] - \Pr[\mathcal{B} \text{ outputs 'i'} \,|E_L(R_x)]| \leq \varepsilon_2.$$

Note that the above specification indicates that E should withstand a chosen-ciphertext attack in the pre-processing mode in the terminology of [19] or CCA-I in [3]. Possible implementation of E_L can be done via pseudo-random permutations (which model block-ciphers). See more details on the efficient implementation of F and E in Section 4.1.

Key Assignment: Another critical cryptographic operation performed in the system is the key assignment method, i.e. how a user u derives the keys L_i for the sets S_i such that $u \in S_i$. We now identify an important property the key assignment method in a subset-cover algorithm should possess that will turn out to be sufficient to provide security for the scheme:

Definition 9 *Let $\mathcal{A}$ be a Subset-Cover revocation algorithm that defines a collection of subsets $S_1, \ldots, S_w$. Consider a feasible adversary $\mathcal{B}$ that*

1. *Selects i, $1 \leq i \leq w$*
2. *Receives the I_u's (secret information that u receives) for all $u \in \mathcal{N} \setminus S_i$*

We say that $\mathcal{A}$ satisfies the key-indistinguishability property *if the probability that $\mathcal{B}$ distinguishes L_i from a random key R_{L_i} of similar length is negligible and we denote this by ε_3, i.e.*

$$|\Pr[\mathcal{B} \textit{ outputs 'i'} \,|L_i] - \Pr[\mathcal{B} \textit{ outputs 'i'} \,|R_{L_i}]| \leq \varepsilon_3.$$

Note that all "information theoretic" key assignment schemes, namely schemes in which the keys to all the subsets are chosen independently, satisfy Definition 9 with $\varepsilon_3 = 0$.

The next lemma is a consequence of the *key-indistinguishability* property and will be used in the proof of Theorem 12, the Security Theorem.

[9] One actually has to repeat such an equivalence proof for the adversaries in question.

Lemma 10 *For any* $1 \leq i \leq w$ *let* $S_{i_1}, S_{i_2} \ldots, S_{i_t}$ *be all the subsets that are contained in* S_i*; Let* $L_{i_1}, \ldots, L_{i_t}$ *be their corresponding keys. For any adversary* $\mathcal{B}$ *that selects* i, $1 \leq i \leq w$, *and receives* I_u *for all* $u \in \mathcal{N} \setminus S_i$, *if* $\mathcal{B}$ *attempts to distinguish the keys* $L_{i_1}, \ldots, L_{i_t}$ *from random keys* $R_{L_{i_1}}, \ldots, R_{L_{i_t}}$ *(of similar lengths) then*

$$\left|\Pr[\mathcal{B} \text{ outputs } \text{`}i\text{'} \,|L_{i_1}, \ldots, L_{i_t}] - \Pr[\mathcal{B} \text{ outputs } \text{`}i\text{'} \,|R_{L_{i_1}}, \ldots, R_{L_{i_t}}]\right| \leq t \cdot \varepsilon_3.$$

Proof: Let us rename the subsets $S_{i_1}, S_{i_2} \ldots, S_{i_t}$ as $S_1, S_2, \ldots S_t$ and order them according to their size; that is for all $j = 1, \ldots, t, S_j \subseteq S_i$ and $|S_1| \geq |S_2| \geq \ldots |S_t|$. We will now use a hybrid argument: consider an "input of the j^{th} type" as one where the first j keys are the true keys and the remaining $t - j$ keys are random keys. $\forall 1 \leq j \leq t$, let p_j be the probability that $\mathcal{B}$ outputs 'i' when challenged with an input of the j^{th} type, namely

$$p_j = \Pr[\mathcal{B} \text{ outputs } \text{`i'} \,|L_1, \ldots, L_j, R_{L_{j+1}}, \ldots, R_{L_t}]$$

Suppose that the lemma doesn't hold, that is $|p_t - p_0| > t \cdot \varepsilon_3$. Hence there must be some j for which $|p_j - p_{j-1}| > \varepsilon_3$. We now show how to create an adversary $\mathcal{B}'$ that can distinguish between R_{L_j} and L_j with probability $> \varepsilon_3$, contradicting the key-indistinguishability property. The actions of $\mathcal{B}'$ result from a simulation of $\mathcal{B}$:

- When $\mathcal{B}$ selects S_i, $\mathcal{B}'$ selects the subset $S_j \subseteq S_i$ from the above discussion (that is, the j for which $|p_j - p_{j-1}| > \varepsilon_3$). It receives I_u for all $u \in \mathcal{N} \setminus S_j$ and hence can provide $\mathcal{B}$ with I_u for all $u \in \mathcal{N} \setminus S_i$.
- When $\mathcal{B}'$ is given a challenge X and needs to distinguish whether X is R_{L_j} or L_j, it creates a challenge to $\mathcal{B}$ that will be $L_1, \ldots, L_j, R_{L_{j+1}}, \ldots, R_{L_t}$ or $L_1, \ldots, L_{j-1}, R_{L_j}, R_{L_{j+1}}, \ldots, R_{L_t}$. Note that due their order $S_1, \ldots, S_{j-1} \not\subset S_j$; since $\mathcal{B}'$ received I_u for all $u \in \mathcal{N} \setminus S_j$ it knows the keys $L_1, \ldots, L_{j-1}$, while $R_{L_{j+1}}, \ldots, R_{L_t}$ are chosen at random. The jth string in the challenge is simply X (the one $\mathcal{B}'$ received as a challenge.) $\mathcal{B}'$ response is simply $\mathcal{B}$'s answer to the query.

The advantage that $\mathcal{B}'$ has in distinguishing between R_{L_j} and L_j is exactly the advantage B' has in distinguishing between $L_1, \ldots, L_j, R_{L_{j+1}}, \ldots, R_{L_t}$ and $L_1, \ldots, L_{j-1}, R_{L_j}, R_{L_{j+1}}, \ldots, R_{L_t}$, which is by assumption larger than ε_3, contradicting the key-indistinguishability property. □

6.2 Security Definition of a Revocation Scheme

To define the security of a revocation scheme we first have to consider the power of the adversary in this scenario (and make pessimistic assumption on its ability). The adversary can pool the secret information of several users, and it may have some influence on the the choice of messages encrypted in this scheme (chosen plaintext). Also it may create bogus messages and see how legitimate users (that will not be revoked) react. Finally to say that the adversary has broken the scheme means that when the users who have provided it their secret information are all revoked (otherwise it is not possible to protect the plaintext) the adversary can still learn something about the encrypted message. Here we define "learn" as distinguishing its encryption from random (again this is equivalent to semantic security).

Definition 11 *consider an adversary* $\mathcal{B}$ *that gets to*

1. *Select adaptively a set $\mathcal{R}$ of receivers and obtain I_u for all $u \in \mathcal{R}$. By adaptively we mean that $\mathcal{B}$ may select messages $M_1, M_2 \ldots$ and revocation set $\mathcal{R}_1, \mathcal{R}_2, \ldots$ (the revocation sets need not correspond to the actual corrupted users) and see the encryption of M_i when the revoked set is $\mathcal{R}_i$. Also $\mathcal{B}$ can create a ciphertext and see how any (non-corrupted) user decrypts it. It then asks to corrupt a receiver u and obtains I_u. This step is repeated $|\mathcal{R}|$ times (for any $u \in \mathcal{R}$).*

2. *Choose a message M as the challenge plaintext and a set $\mathcal{R}$ of revoked users that must include all the ones it corrupted (but may contain more).*

$\mathcal{B}$ then receives an encrypted message M' with a revoked set $\mathcal{R}$. It has to guess whether $M' = M$ or $M' = R_M$ where R_M is a random message of similar length. We say that a revocation scheme is secure if, for any (probabilistic polynomial time) adversary $\mathcal{B}$ as above, the probability that $\mathcal{B}$ distinguishes between the two cases is negligible.

6.3 The Security Theorem

We now state and prove the main security theorem, showing that the key-indistinguishability property is sufficient for a scheme in the subset-cover framework to be secure in the sense of Definition 11. Precisely,

Theorem 12 *Let $\mathcal{A}$ be a Subset-Cover revocation algorithm where the key assignment satisfies the key-indistinguishability property (Definition 9) and where E and F satisfy the above requirements. Then $\mathcal{A}$ is secure in the sense of Definition 11 with security parameter $\delta \leq \varepsilon_1 + 2mw(\varepsilon_2 + 4w\varepsilon_3)$, where w is the total number of subsets in the scheme and m is the maximum size of a cover.*

Proof: Let $\mathcal{A}$ be a Subset-Cover revocation algorithm with the key indistinguishability property. Let $\mathcal{B}$ be an adversary that behaves according to Definition 11, where δ is the probability that $\mathcal{B}$ distinguishes between an encryption of M and an encryption of a random message of similar length.

Recall that the adversary adaptively selects a set of receivers $\mathcal{R}$ and obtains I_u for all $u \in \mathcal{R}$. $\mathcal{B}$ then selects a challenge message M. Let $\mathcal{S} = S_{i_1}, S_{i_2}, \ldots S_{i_m}$ be the cover of $\mathcal{N} \setminus \mathcal{R}$ defined by $\mathcal{A}$. As a challenge, $\mathcal{B}$ then receives an encrypted message and is asked to guess whether it encrypts M or a random message R_M of the same length as M. We consider $\mathcal{B}$'s behavior in case not all the encryptions are proper. Let a "ciphertext of the j^{th} type" be one where the first j subsets are noisy and the remaining subsets encode the correct key. In other words the body is the encryption using F_K and the header is:

$$[i_1, i_2, \ldots, i_m, E_{L_{i_1}}(R_K^1), E_{L_{i_2}}(R_K^2), \ldots, E_{L_{i_j}}(R_K^j), E_{L_{i_{j+1}}}(K), \ldots, E_{L_{i_m}}(K)]$$

where K is a random key and $\{R_K^i\}$ are random strings of the same length as the key K. Let Δ_j be the advantage that for a ciphertext of the j^{th} type $\mathcal{B}$ distinguishes between the cases where $F_K(M)$ or $F_K(R_M)$ are the body of the message. I.e.

$$\Delta_j = |\Pr[\mathcal{B} \text{ outputs 'i' } | \text{body is } F_K(M)] - \Pr[\mathcal{B} \text{ outputs 'i' } | \text{body is } F_K(R_M)]|,$$

where the header is of the j^{th} type.

The assumption that $\mathcal{B}$ can break the revocation system implies that $\Delta_0 = \delta$. We also know that $\Delta_m \leq \varepsilon_1$, the upper bound on the probability of breaking F_K, since in ciphertexts of the m^{th} type the encryptions $E_{L_{i_j}}$ in the header contain no information on the key K used for the body so K looks random to $\mathcal{B}$. Hence there must be some $0 < j \leq m$ such that

$$|\Delta_{j-1} - \Delta_j| \geq \frac{\delta - \varepsilon_1}{m}.$$

For this j it must be the case that for either M or R_M the difference in the probability that $\mathcal{B}$ outputs 'i' between the case when the header is of the j^{th} type and when it is of the $(j-1)^{th}$ type (and the same message is in the body) is at least $\frac{\delta-\varepsilon_1}{2m}$.

A ciphertext of the $(j-1)^{th}$ type is noticeably different from a ciphertext of the j^{th} type only if it is possible to distinguish between $E_{L_{i_j}}(K)$ and $E_{L_{i_j}}(R_K)$. Therefore, the change in the distinguishing advantage $|\Delta_{j-1} - \Delta_j| \geq \frac{\delta-\varepsilon_1}{m}$ can be used to either break the encryption E_L or to achieve an advantage in distinguishing the keys. We will now show how $\mathcal{B}$ can be used to construct an adversary $\mathcal{B}'$ that either breaks E_L or breaks the key-indistinguishability property, as extended by Lemma 10. This in turn is used to derive bounds on δ.

Formally, we now describe an adversary $\mathcal{B}'$ that will use $\mathcal{B}$ as follows.

- $\mathcal{B}'$ picks at random $1 \leq i \leq w$ and asks to obtain I_u for all $u \notin S_i$; this is a guess that $S_{i_j} = S_i$.
- $\mathcal{B}'$ receives either $L_0, L_1, \ldots, L_t$ or $R_{L_0}, R_{L_1}, \ldots, R_{L_t}$ where $L_0 = L_i$, the key of the subset S_i, and $L_1, \ldots, L_t$ are defined as the keys in Lemma 10. It attempts to distinguish between the case where the input corresponds to true keys and the case where the input consists of random keys.
- $\mathcal{B}'$ simulates $\mathcal{B}$ as well as the Center that generates the ciphertexts and uses $\mathcal{B}$'s output:
  - When the Center is faced with the need to encrypt (or decrypt) using the key of subset S_j such that $S_j \not\subseteq S_i$, then it knows at least one $u \in S_j$; from I_u it is possible to obtain L_j and encrypt appropriately. If $S_j \subseteq S_i$ then $\mathcal{B}'$ uses the key that was provided to it (either L_j or R_{L_j}).
  - When $\mathcal{B}$ decides to corrupt a user u, if $u \notin S_i$, then $\mathcal{B}'$ can provide it with I_u. If $u \in S_i$ then the guess that $i_j = i$ was wrong and we abort the simulation.
  - When the Center needs to generate the challenge ciphertext M for $\mathcal{B}$, $\mathcal{B}'$ finds a cover for $\mathcal{R}$, the set of users corrupted by $\mathcal{B}$. If $i_j \neq i$, then the guess was wrong and the simulation is aborted. Otherwise a random key K is chosen and a body of a message encrypted with K is generated where the encrypted message is either M or R_M (depending to whom the difference between Δ_j and Δ_{j-1} is due) and one of two experiments is performed:

    Experiment j: Create a header of ciphertext of the j^{th} type.

    Experiment $j-1$: Create a header of ciphertext of the $(j-1)^{th}$ type.

    Provide as challenge to $\mathcal{B}$ the created header and body.
- If the simulation was aborted, output 'i' or 'ii' at random. Otherwise provide $\mathcal{B}$'s output.

Denote by P_L^j (and P_L^{j-1} resp.) the probability that in experiment j (experiment $j-1$) in case the input to $\mathcal{B}'$ are the true keys the simulated $\mathcal{B}$ outputs 'i'; denote by P_R^j (and P_R^{j-1} resp.) the probability that in experiment j (experiment $j-1$) in case the input to $\mathcal{B}'$ are random keys the simulated $\mathcal{B}$ outputs 'i'. We claim that the differences between all these 4 probabilities can be bounded:

Claim 13 $|P_L^j - P_L^{j-1}| \geq \frac{\delta-\varepsilon_1}{2wm}$

Proof: In case the input to $\mathcal{B}'$ are the true keys, the resulting distribution that the simulated $\mathcal{B}$ experiences is what it would experience in a true execution (where the difference between a j^{th} ciphertext and $(j-1)^{\text{th}}$ ciphertext are at least $\frac{\delta-\varepsilon_1}{2m}$). The probability that the guess was correct is $1/w$ and this is independent of the action of $\mathcal{B}$, so we can experience a difference of at least $\frac{\delta-\varepsilon_1}{2wm}$ between the two cases. □

Claim 14 $|P_R^j - P_R^{j-1}| \leq \varepsilon_2$

Proof: Since otherwise we can use $\mathcal{B}'$ to attack E: whenever there is a need to use the key corresponding to the set S_i, ask for an encryption using the random key. Similarly use the K in the challenge for E as the one in the challenge of $\mathcal{B}'$. □

Claim 15 $|P_R^j - P_L^j| \leq w \cdot \varepsilon_3$ *and* $|P_R^{j-1} - P_L^{j-1}| \leq w \cdot \varepsilon_3$

Proof: If any of the two inequalities does not hold, then we can use $\mathcal{B}'$ as an adversary for Lemma 10 and contradict the safety of the key assignment (we know that $t \leq w$). □

From these three claims and applying the inequality $|a-b| - |c-d| \leq |a-c| + |b-d|$ we can conclude that

$$\frac{\delta - \varepsilon_1}{2wm} - \varepsilon_2 \leq 2w \cdot \varepsilon_3$$

and hence the overall security parameter of $\mathcal{A}$ satisfies $\delta \leq \varepsilon_1 + 2mw(\varepsilon_2 + 2w\varepsilon_3)$.

□

Weaker notions of security It is interesting to deal with the case where the encryption provided by F is not so strong. To combat copyright piracy it may not make sense to protect a *specific* ciphertext so that breaking it is very expensive; on the other hand we do want to protect the long lived keys of the system. The security definition (Definition 11) can easily be adapted to the case where distinguishing $F_K(M)$ from $F_K(R_M)$ cannot be done in some time T_1 where T_1 is not too large (this may correspond to using a not very long key K): the challenge following the attack is to distinguish $F_K(M)$ from $F_K(R_M)$ it time less than T_1' not much smaller than T_1. Essentially the same statement and proof of security as Theorem 12 hold. The fact that retrieving K does not have to be intractable, just simply expensive, means that K does not necessarily have to be long; see discussion on the implications on the total message length in Section 4.1.

It is also possible to model the case where the protection that F_K provides is not indistinguishability (e.g. F_K encrypts only parts of the message M that are deemed more important). In this case we should argue that the header does not provide more information regarding M than does $F_K(M)$. More precisely, suppose that $\mathcal{M}$ is a distribution on messages M and let $\mathcal{B}$ be an adversary that attacks the system as in Definition 11 but is given as a challenge a valid encryption of a message $M \in_R \mathcal{M}$ and attempts to compute some function of M (e.g. M defines a piece of music and the function is to map it to sounds). A scheme is considered secure if for any $\mathcal{M}$ and $\mathcal{B}$ there is a $\mathcal{B}'$ that simply receives $F_K(M)$ without the header and (i) performs an amount of work proportional to $\mathcal{B}$ after receiving the challenge and (ii) whose output is indistinguishable from $\mathcal{B}$'s output; the distinguisher should have access to M. Here again for any subset cover algorithm where E and the key assignment algorithm satisfy the requirements of Section 6.1 the resulting scheme will satisfy the relaxed definition.

Acknowledgements

We thank Omer Horvitz for many comments regarding the paper and the implementation of the system. We thank Ravi Kumar and Florian Pestoni for useful comments.

References

[1] N. Alon and J. Spencer, **The Probabilistic Method**, John Wiley & Sons, 1992.

[2] J. Anzai, N. Matsuzaki and T. Matsumoto, A Quick Group Key Distribution Sceheme with "Entity Revocation". Advances in Cryptology - Asiacrypt '99, Lecture Notes in Computer Science 1716, Springer, 1999, pp. 333–347.

[3] M. Bellare, A. Desai, D. Pointcheval, P. Rogaway: *Relations Among Notions of Security for Public-Key Encryption Schemes*, Advances in Cryptology - CRYPTO'98, Lecture Notes in Computer Science 1462, Springer, 1998, pp. 26–45.

[4] M. Bellare, A. Desai, E. Jokipii and P. Rogaway, *A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation,* Proc. of 38th IEEE Symposium on Foundations of Computer Science, 1997, pp. 394–403.

[5] O. Berkman, M. Parnas and J. Sgall, Efficient Dynamic Traitor Tracing. Proc. of the 11th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 586–595, 2000.

[6] M. Blum and S. Micali, How to Generate Cryptographically Strong Sequences of Pseudo Random Bits, SIAM J. Compt., Vol. 13, pp. 850–864, 1984.

[7] D. Boneh and M. Franklin. An efficient public key traitor tracing scheme. Advances in Cryptology - Crypto '99, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, pp. 338–353, 1999.

[8] D. Boneh and M. Franklin. Identity Based Encryption. Manuscript, 2001.

[9] D. Boneh, and J. Shaw. *Collusion Secure Fingerprinting for Digital Data,* IEEE Transactions on Information Theory, Vol 44, No. 5, pp. 1897–1905, 1998.

[10] A. Brodnick and J. I. Munro, Membership in Constant Time and Almost-Minimum Space. SIAM J. Comput. 28(5), 1999, pp. 1627–1640.

[11] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, A. Sahai. Exposure-Resilient Functions and All-or-Nothing Transforms. EUROCRYPT 2000, pp. 453–469.

[12] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor and B. Pinkas, Multicast Security: A Taxonomy and Some Efficient Constructions. Proc. of INFOCOM '99, Vol. 2, pp. 708–716, New York, NY, March 1999.

[13] R. Canetti, T. Malkin, K. Nissim, Efficient Communication-Storage Tradeoffs for Multicast Encryption. EUROCRYPT 1999: pp. 459–474.

[14] R. Cramer and V. Shoup, A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. Advances in Cryptology - CRYPTO 1999, Lecture Notes in Computer Science 1462, Springer, pp. 13–25.

[15] Z. J. Czech, G. Havas and B. S. Majewski, *Perfect Hashing*, Theoretical Computer Science 182, 1997, 1–143.

[16] B. Chor, A. Fiat and M. Naor, Tracing traitors Advances in Cryptology - CRYPTO '94, Lecture Notes in Computer Science, Vol. 839, Springer, pp. 257–270, 1994.

[17] B. Chor, A. Fiat, M. Naor and B. Pinkas, Tracing traitors. IEEE Transactions on Information Theory, Vol. 46, No. 3, May 2000.

[18] Content Protection for Recordable Media http://www.4centity.com/4centity/tech/cprm

[19] D. Dolev, C. Dwork and M. Naor, *Nonmalleable Cryptography,* SIAM J. Computing 30(2), 2000, pp. 391–437.

[20] C. Dwork, J. Lotspiech and M. Naor, Digital Signets: Self-Enforcing Protection of Digital Information, 28th Symposium on the Theory of Computation (1996), pp. 489– 498.

[21] P. Erdos and R. Rado, Intersection Theorems for Systems of Sets. Journal London Math. Soc. 35 (1960), pp. 85–90.

[22] U. Feige, P. Raghavan, D. Peleg, E. Upfal, Computing with Noisy Information. SIAM J. Comput. 23(5): 1001-1018 (1994)

[23] A. Fiat and M. Naor, *Broadcast Encryption*, Advances in Cryptology - CRYPTO '93, Lecture Notes in Computer Science 773, Springer, 1994, pp. 480—491.

[24] A. Fiat and T. Tassa, *Dynamic Traitor Tracing* Advances in Cryptology - CRYPTO '99, Lecture Notes in Computer Science, Vol. 1666, 1999, pp. 354–371.

[25] E. Fujisaki and T. Okamoto, *Secure Integration of Asymmetric and Symmetric Encryption Schemes*, Advances in Cryptology - CRYPTO 1999, Lecture Notes in Computer Science, Vol. 1666, 1999, pp. 537–554.

[26] E. Gafni, J. Staddon and Y. L. Yin, *Efficient Methods for Integrating Traceability and Broadcast Encryption*, Advances in Cryptology - CRYPTO 1999, Lecture Notes in Computer Science, vol. 1666, Springer, 1999, pp. 372–387.

[27] J.A. Garay, J. Staddon and A. Wool, Long-Lived Broadcast Encryption. Advances in Cryptology - CRYPTO'2000, Lecture Notes in Computer Science, vol 1880, pp. 333–352, 2000.

[28] O. Goldreich, S. Goldwasser and S. Micali, How to Construct Random Functions. JACM 33(4): 792–807 (1986)

[29] S. Goldwasser and S. Micali. *Probabilistic Encryption*, Journal of Computer and System Sciences, Vol. 28, April 1984, pp. 270–299.

[30] R. Kumar, R. Rajagopalan and A. Sahai, Coding Constructions for blacklisting problems without Cop-mutational Assumptions. Advances in Cryptology - CRYPTO '99, Lecture Notes in Computer Science, vol 1666, 1999, pp. 609–623.

[31] M. Luby and J. Staddon, Combinatorial Bounds for Broadcast Encryption. Advances in Cryptology - EUROCRYPT '98, Lecture Notes in Computer Science, vol 1403, 1998, pp. 512–526.

[32] D. McGrew, A. T. Sherman, "Key Establishment in Large Dynamic Groups Using One-Way Function Trees", submitted to IEEE Transactions on Software Engineering (May 20, 1998).

[33] Moni Naor, *String Matching with Preprocessing of Text and Pattern*, ICALP 1991 Proceeding, Lecture Notes in Computer Science, Vol. 510, Springer, 1991, pp. 739–750.

[34] M. Naor and B. Pinkas, Threshold traitor tracing, Advances in Cryptology - Crypto '98, Lecture Notes in Computer Science, Vol. 1462, pp. 502–517.

[35] M. Naor and B. Pinkas, Efficient Trace and Revoke Schemes Financial Cryptography '2000, Lecture Notes in Computer Science, Springer.

[36] M. Naor, B. Pinkas and O. Reingold, *Distributed Pseudo-random Functions and KDCs,* Advances in Cryptology -EUROCRYPT 1999, Lecture Notes in Computer Science, vol. 1592 Springer, 1999, pp. 327–346.

[37] M. Naor and O. Reingold, *Number-theoretic Constructions of Efficient Pseudo-random Functions*, Proc. of 38th IEEE Symposium on Foundations of Computer Science, 1997, 458–467.

[38] R. Pagh, Low redundancy in static dictionaries with O(1) lookup time, Proceedings of ICALP '99, Lecture Notes in Computer Science 1644, Springer, 1999, pp. 595–604.

[39] B. Pfitzmann, Trials of Traced Traitors, Information Hiding Workshop, First International Workshop, Cambridge, UK. Lecture Notes in Computer Science, Vol. 1174, Springer, 1996, pp. 49–64.

[40] R. L. Rivest All-or-Nothing Encryption and the Package Transform. Proc. 4th Fast Software Encryption International Workshop, 1997, Lecture Notes in Computer Science, Vol. 1267, Springer, 1997, pp. 210–218.

[41] R. Safavi-Naini and Y. Wang, Sequential Traitor Tracing Advances in Cryptology - CRYPTO 2000, Lecture Notes in Computer Science, vol 1880, pp. 316–332, 2000.

[42] D.M. Wallner, E.J. Harder and R.C. Agee Key Management for Multiast: Issues and Architectures, IETF draft wallner-key, July 1997. ftp://ftp.ietf.org/internet-drafts/draft-wallner-key-arch-01.txt.

[43] C. K. Wong, M. Gouda and S. Lam, Secure Group Communications Using Key Graphs, SIGCOMM 1998.

Individual Authentication in Multiparty Communications

Francesco Bergadano and Davide Cavagnino
University of Turin,

and
Bruno Crispo
Cryptomathic Italia

In this paper we introduce a new authentication scheme to achieve individual authentication in group communications. The scheme is particularly efficient and suitable for applications where users require to transmit stream of data of undefined length through noisy channels. Our scheme is in fact, robust against loss of packets during the transmission. We present the scheme called chained stream authentication (CSA) and then we prove that the scheme is conditionally secure. We then describe two variations of CSA, one interactive to use when multicast is available and a timed version suitable for broadcast communications. We conclude by describing our implementation of the timed version that is integrated and fully compatible with RAT.

Categories and Subject Descriptors: D.4.6 [**Security and Protection**]: Authentication; H.4.3 [**Communications Applications**]: Teleconferencing

General Terms: Security, Authentication, Multicast, Protocols

Additional Key Words and Phrases: Multicast transmission, individual authentication, teleconferencing

---

Name: Francesco Bergadano
Affiliation: Computer Science Department, University of Turin
Address: Corso Svizzera 185 - 10149 Torino - Italia, E-mail: Francesco.Bergadano@di.unito.it
Name: Davide Cavagnino
Affiliation: Computer Science Department, University of Turin
Address: Corso Svizzera 185 - 10149 Torino - Italia, E-mail: Davide.Cavagnino@di.unito.it
Name: Bruno Crispo
Affiliation: Cryptomathic Italia
Address: Corso Svizzera 185 - 10149 Torino - Italia, E-mail: crispo@cryptomathic.com

1. INTRODUCTION

The networks of the future will be able to support gigabit bandwidths for large groups of users. These users will possess various qualities of service options and multimedia applications that include video, voice, and data, all on the same network backbone. The desire to create small groups of users all interconnected and capable of communicating with each other, but who are securely isolated from all other users on the network is being expressed strongly in a variety of communities. Today most network applications are based upon the client-server paradigm and make use of unicast (peer-to-peer) packet delivery. However, many emerging applications (e.g., wargaming, law enforcement, teleconferencing, command and control conferencing, disaster relief, and distributed computing, collaborative work, etc.) are based upon a group communication model. That is, they require packet delivery from one or more authorized sender(s) to a variable large number of authorized receivers. While peer-to-peer security is a mature and well developed field, secure group communication remains relatively unexplored. Contrary to a common initial impression, secure group communication is not a simple extension of secure two-party communication. There are two important differences. First, protocol efficiency is of greater concern due to the number of participants and to the distances among them. The second difference is due to group dynamics. Two-party communication can be viewed as a discrete phenomenon: it starts, lasts for a while and normally ends. Group communication is usually more complicated, it starts, the group mutates (members leave and join) and there might not be a well-defined end. This complicates security services among which group key management (i.e., initializing the secure group with a common net key, rekeying the group, etc.) is the most important.

In group communications, data authentication comes in two different kinds. In one case, integrity and authentication of origin with respect to a specific individual sender are guaranteed. We then speak of **individual authentication**. In the other case, data is known to have been sent and/or modified by a member of a specified group - we do not know *which* member, and actually any group member could even have modified the data while in transit. We then speak of **group authentication**.

Individual authentication is especially important in broadcast applications. In this case, there is usually one main transmitter and a possibly large, or very large, number of receivers. This makes group authentication of little use - we want to know that the broadcast data comes from the designated transmitter; receivers cannot be trusted and usually they do not need to be so because they just receive the information.

This paper addresses individual authentication. A general authentication protocol will be obtained, that is secure even when attackers may adaptively choose authenticated data streams. For an interactive scenario, that is typical of multicast conferencing, the proposed solution is substantially more efficient than in previous approaches.

This paper is organised as follows: section 2 reviews the works on individual authentication in group communication, section 3 presents the chained stream authentication (CSA) protocol and discusses the concept of security against continuations. The two following sections present two variants of CSA, one for interacting

parties (section 4) and one for broadcast communication (section 5). Then, section 6 discusses the implementation of the presented protocol in a real application, and section 7 draws some conclusions.

2. RELATED WORK

Individual authentication is not commonly found in broadcast and multicast, mainly because it is technically difficult or inefficient. There are two obvious solutions. First, digital signatures could be used for each block of data being broadcast. This works, and is secure, but is inefficient. Especially for receivers that may be equipped only with simple hardware, or even with set-top-box solutions, this may be a major difficulty. Furthermore, in these applications the receiving hardware is already busy with multimedia processing and cannot spend precious time and resources in authentication. In teleconferences this may also apply to senders. Moreover, digital signatures bring non-repudiation, which may not be desired or necessary in some cases. Second, symmetric message authentication codes (MACs), which are a lot faster, could be used and should be of no concern from the point of view of efficiency. But then the broadcaster needs a different shared key for each receiver. This causes two problems: (1) every block needs to be authenticated with as many MACs as there are receivers - which is very inconvenient, e.g., for TV-like broadcast, and (2) key management is very complex.

In the literature, a number of solutions have been proposed that may be adequate in some contexts. With the choice of symmetric authentication, it is possible to use a fixed number $K < N$ of MAC keys, that does not depend on the size N of the multicast group [Canetti and Pinkas 1998; Dyer et al. 1995]. The sender knows K keys, and each receiver knows $K/2$ keys, chosen at random and distributed by a key distribution center. Each stream block is authenticated with K MACs, corresponding to the K keys, and receivers check the MACs for which they have a key. Obviously, receiver collusion can lead to forged individual authentication codes. More work is available for the choice of efficient digital signatures. Online/offline signatures [Even et al. 1989] may be used to split the computation into an expensive offline phase, and an efficient online phase performed when the data becomes available. Part of the signature can be performed by a signature server, without compromising authentication or non-repudiation [Asokan et al. 1996]. One time-signatures [Bleichenbacher and Maurer 1996; Lamport 1979; Merkle 1987] are an efficient alternative to be used for stream authentication. Gennaro and Rohatgi [Gennaro and Rohatgi 1997], have proposed an efficient stream signing method that is based on a chain of one-time signatures. Only the first block is signed with a standard digital signature, and includes a one-time public key to be used in the second step. The corresponding one-time secret key is used to sign the second block, together with a one-time public key used in the third step. The process is continued until the end of the stream and real-time authentication and non-repudiation of every stream prefix is guaranteed. The disadvantage of this approach lies in the length of the authentication information. However this length does not depend on the number of participants, and the technique scales up well to very large multicast groups and even broadcast applications. A similar scheme has also been applied to authentication in routing protocols [Zhang 1998].

The problem of lengthy authentication information is solved by the Guy Fawkes

protocol [Anderson et al. 1998] which uses hash functions to bind a sequence of events to an initial value and to guarantee the sequence integrity. Establishing by any means (i.e., a digital signature) the identity of the entity that knows the initial value allows to secure authentication of all the subsequent events. Thus, the sender first commits to a string of the form $Z_i = \{M_i, X_i, h(X_{i+1})\}$, where M_i denotes message i, X_i stands for a random number used as codeword. This commitment binds the message to the codeword and its successor. The sender then reveals the value of this string, proving her knowledge of the codeword and thus authenticating herself. To avoid a man-in-the-middle attack by simply intercepting Z_{i+1} and the following message that reveals the codeword and by doing so being able to insert, without been detected by the recipient, a forged message in the stream, the first random secret number (codeword) needs to be bootstrapped by using, for example, a digital signature.

Guy Fawkes was an important milestone but suffered from two major limitations. First, the sender need to be acknowledged about the reception of the packet by the receiver before she can send the next packet. This requirement fits well protocols like TCP but it is expensive to implement in protocols like ICMP that do not require acknowledgement for each packet sent. Publication of a key before due acknowledgement allows an opponent to forge the rest of the stream. Second, the protocol does not tolerate packet loss.

Cheung [Cheung 1997] uses a method very similar to Guy Fawkes to efficiently authenticate link state updates (LSU) that are periodically distributed to routers. In this application the message is always an LSU that describes the status of the links incident to the routers. The protocol uses an optimistic approach in the sense that the LSU is immediately accepted as valid when is received by the router and after a tuneable but fixed delay the correspondent key is expected so that the accepted LSU can be actually validated. The method does not distinguish between forged LSUs and the case where the packet containing the validation key is simply lost but the correspondent LSU is genuine. This is an important limitation because the losses of packets in large networks are very likely to happen. Besides, this method requires some degree of synchronization among all routers that is not easy to achieve in practice. The author does not provide any assurance about the correctness of the scheme.

More recently Wong and Lam in [Wong and Lam 1998], introduced a chaining technique based on Merkle trees [Merkle 1987], suitable to authenticate in efficient way both real-time and non real-time streams. Their approach is in some respect similar to ours, but we use as primitive the hash-chains introduced by Lamport [Lamport 1979] instead. Similarly to us, they use a single digital signature for a block of packets. However their signing operation even if delay-bounded requires a higher delay compared to our signing operation because we do not need to build a tree before starting to compute packet digests but each packet digest is independent from the others [Bergadano et al. ; Bergadano et al. 2000a; Bergadano et al. 2000b]. Our verification procedure is also more efficient. We need to compute a constant number of packet digests for each verification (2) while they need to compute $O(\log(d))$ digests, where d is the height of the tree. Another very important difference between all the solutions we have presented so far and our scheme is that they are not robust. In this scheme, if a packet is lost the authentication of all the

remaining packets of the tree that have not yet been authenticated is lost. With our algorithm if we lose a packet only a single packet of data will be affected, furthermore our chaining technique can continue without disruption and can guarantee authentication of the following packets of the chain.

Recently Perrig et al. [Perrig et al. 2000] introduced two schemes TESLA and EMSS that have been developed from ideas introduced in the Guy Fawkes protocol [Anderson et al. 1998]. The first of their schemes, TESLA is very similar to one of our solutions.

To the best of our knowledge, all these groups have been working independently.

3. CHAINED STREAM AUTHENTICATION (CSA)

A stream is a long sequence (of undefined length) of bits that a sender sends to one or more receivers. A stream is different from a message in the sense that it must be processed by the receivers at almost the same rate it is received. Examples of digital streams are the video and audio sent over the Internet.

This section presents an authentication protocol for entities that continuously exchange data. Then, its security properties are shown.

Following the formalization of [Goldwasser et al. 1988] and [Gennaro and Rohatgi 1997], we give the following definitions:

Definition 1. [Negligibility]: define a *security parameter*, n, and say that a function $\epsilon(n)$ is "negligible", if, for all constants c, there is n_0 such that, for $n > n_0$, $\epsilon(n) < 1/n^c$.

Definition 2. [Signature scheme]: a signature scheme is a triple (G, Sig, V) of probabilistic polynomial time algorithms, where (1) G is used to generate a key pair (SK, PK), where SK is the private key and PK is the public key, (2) Sig is used to sign any message M, using the secret key SK, and (3) V is the signature verification algorithm, such that V(PK, M, Sig(SK, M)) = 1, for any message M.

We will use a signature scheme that is secure against adaptively chosen message attacks [Goldwasser et al. 1988]: the probability of forging a signature is negligible, even when a signature oracle is available.

Similarly, we give the following

Definition 3. [Stream authentication scheme]: a *stream authentication scheme* is a triple of probabilistic polynomial-time algorithms (GA, AA, VA), where

—On input 1^n, GA outputs a pair of keys (SK, PK)$\in \{0,1\}^{2n}$.

—AA is the authentication algorithm, and receives in input a secret key SK, and a stream $S = S_1, S_2, ..., S_i$, consisting of a finite number i of *blocks*. AA outputs an authenticated stream $S' = S'_1, ..., S'_i$, where $S'_j = (S_j, auth_j)$, being $auth_j$ some kind of authentication data.

—The verification algorithm VA is such that VA(PK, AA(SK, S)) = 1. When VA(PK, S') = 1, we will say that S' is *valid*.

Assuming
$|T|$: length of stream T;
$S^{(r)}$: r-th stream in a set of streams $\{S^{(1)}, \ldots, S^{(n)}\}$;
$h^k(\alpha) = h^{k-1}(h(\alpha))$: hash function h applied k times to initial value α;

$MAC_\beta(S)$: Message Authentication Code computed on data S with key β;
SN: session number (comprising the set of identifiers of the participants in the case of I-CSA protocol);
$Sig(SK_A, \alpha, \beta)$: signature performed by A on concatenation of data α and β, where SK_A is the private key of A,
we may now define our new proposed scheme.

Definition 4. **[Chained stream authentication scheme]**: a *chained stream authentication scheme* is a stream authentication scheme where:

—As a generator GA, we use the generator of a signature scheme (G, Sig, V), secure against chosen message attacks.

—The authentication algorithm AA will be called a *Chained Stream Authentication* algorithm (CSA). This algorithm first generates a secret α, computes $h^k(\alpha)$ for some $k > i$, and then produces the following output:
$S'_1 = S_1,\ MAC_{h^{k-1}(\alpha)}(S_1), h^k(\alpha), SN, Sig(SK, h^k(\alpha), SN)$
$S'_2 = S_2,\ MAC_{h^{k-2}(\alpha)}(S_2), h^{k-1}(\alpha)$
....
$S'_i = S_i,\ MAC_{h^{k-i}(\alpha)}(S_i), h^{k-i+1}(\alpha)$
By MAC, we denote a secure Message Authentication Code, e.g. such that the probability of forging a valid code is negligible, even when a MAC oracle is available. For h, we will use a collision resistant hash function. The authentication includes a session number SN that is incremented for every new stream.

—The verification algorithm VA will output 1 if the initial asymmetric signature is valid, if all the MACs are correct, and if the hash chain of the MAC keys is consistent, i.e. the hash of a key produces the previous key in the chain.

3.1 Security against Continuations

We define continuations as follows:

Definition 5. [Continuation of a stream]: A stream S_2 is a **continuation** of a stream S_1, denoted by $S_1 \subset S_2$, if S_1 is a proper prefix of S_2.

The same definition applies to authenticated streams under (GA, AA, VA). A valid authenticated continuation S'_2 of an authenticated stream S'_1 must then be such that $S'_1 \subset S'_2$ and VA(PK, S'_2) = 1.

What are the security properties of the proposed scheme? Clearly, given a stream authentication oracle, it is possible to forge new valid authenticated streams, because the MAC keys become known. Therefore, CSA is not "secure" according to the definition of [Gennaro and Rohatgi 1997], that can be rephrased as follows:

Definition 6. A stream authentication scheme (GA, AA, VA) is secure if any probabilistic polynomial-time algorithm F, given as input the public key PK and adaptively chosen authenticated streams $S'^{(j)}$, outputs a new valid authenticated stream $S' \not\subseteq S'^{(j)}$, for all j, only with negligible probability.

This definition of security obviously does not apply to the CSA scheme: the forger F may ask for just one authenticated stream, change any block but the last, and recompute the corresponding MACs using the available keys.

However, our scheme satisfies a weaker security notion, which we will call "security against continuations". Security against continuations means, informally, that it is unfeasible to produce valid continuations of observed valid streams. This weaker notion will nevertheless be sufficient for building secure authentication protocols, after some means of sender/receiver synchronization is achieved, as described in Sections 4.1 and 4.2. More precisely, security against continuations corresponds to the following:

Intuition 1. A forger may produce a new valid stream S only if it is associated to a stream T, that was used previously. Moreover, if $|S| = |T|$, and the last blocks of S and T are different, then it will be impossible to produce a valid continuation of S.

Next, we formalize the above intuition and prove that it applies to CSA (in Lemma 1).

Definition 7. [Security against continuations]: A stream authentication scheme is **secure against continuations** if there is no polynomial time algorithm F that, given adaptively chosen authenticated streams $S'^{(1)}, ..., S'^{(k)}$, is able to generate a valid authenticated stream $S' = S'_1, ..., S'_j$ with non-negligible probability, unless $\exists i \in [1, k]$ such that $S'^{(i)}_1 = S^{(i)}_1$, MAC^i_1, H, SN, $Sig(SK, H, SN)$, where $S'_1 = S_1$, MAC_1, H, SN, $Sig(SK, H, SN)$, and one of the following holds:

(1) $j < |S'^{(i)}|$, or

(2) $j = |S'^{(i)}|$, and $S'_j = S'^{(i)}_j$, or

(3) $j = |S'^{(i)}|$, and $S'_j \neq S'^{(i)}_j$, and there is no polynomial time algorithm F' that can generate with non-negligible probability a valid continuation $T' \supset S'$, given $S'^{(1)}, ..., S'^{(k)}$, possibly other adaptively chosen authenticated streams, and any valid authenticated continuation of $S'^{(i)}$.

Security against continuations is a complicated notion, but it will lead us to a simple concept of stream authentication in Theorem 1. First, though, we need the following:

LEMMA 1. *Suppose that, in the CSA authentication scheme,*
(1) (G, S, V) is a secure signature scheme,
(2) g is a pseudorandom function,
(3) $h(x) = g_x(0)$ and $MAC_k(x) = g_k(1, x)$ where g concatenates all its parameters to obtain only one,
(4) g is such that h is a collision resistant hash function.
Then, the scheme (GA, CSA, VA) is secure against continuations.

PROOF. (of Lemma 1.)
Suppose that a forger F exists that can produce a valid authenticated stream S' with a non-negligible probability ϵ, contraddicting the thesis. Then, one of the two following cases must hold, and at least one must hold with probability at least $\epsilon/2$:
Case 1 $S'_1 = S_1, MAC_1, H, SN, Sig(SK, H, SN)$ and there is no $S'^{(i)}$ such that $S'^{(i)}_1 = S'^{(i)}_1, MAC^{(i)}_1, H, SN, Sig(SK, H, SN)$.
Then, we can use the forger F to construct algorithm F1 that breaks the asymmetric signature scheme (G, S, V). The constructed algorithm F1 has access to an

oracle for S, and starts by calling F as a subroutine. When F requires an authenticated stream $S'^{(i)}$, F1 generates a random secret $\alpha^{(i)}$, computes $h^k(\alpha^{(i)})$, and asks the oracle for $Sig(SK, h^k(\alpha^{(i)}), SN)$. Then, knowing $\alpha^{(i)}$, F1 authenticates the rest of the stream as required by CSA, and outputs the authenticated stream $S'^{(i)}$ for F. The process continues until, with probability at least $\epsilon/2$, F outputs the new valid authenticated stream S'. In this case (Case 1), S' must be such that $S'_1 = S_1, MAC_1, H, SN, Sig(SK, H, SN)$ and there is no $S'^{(i)}$ generated by F1 such that
$S'^{(i)}_1 = S'^{(i)}_1, MAC^{(i)}_1, H, SN, Sig(SK, H, SN)$.

This means that a signature $Sig(SK, H, SN)$ was never queried by F1 to the oracle. Hence F1 breaks (S, G, V) by outputting the new valid signature $H, SN, Sig(SK, H, SN)$.

Case 2 $S'_1 = S_1, MAC_1, H, SN, Sig(SK, H, SN)$ and there is $S'^{(i)}$ such that $S'^{(i)}_1 = S'^{(i)}_1, MAC^{(i)}_1, H, SN, Sig(SK, H, SN)$. Then there are three cases, and one must hold with probability at least $\epsilon/6$:

Case 2.1 $|S'| < |S'^{(i)}|$. This case does not contraddict the Lemma.

Case 2.2 $|S'| > |S'^{(i)}|$. In this case we can construct F2 that can invert h, using the forger F, and hence break g. The inverter F2 is given a value α and must compute $h^{-1}(\alpha)$ with non-negligible probability. F2 calls GA to generate a key pair (SK, PK), and then runs F as a subroutine. Let SN_{max} be the maximum number of authenticated streams that F will ask for. Clearly SN_{max} must be polynomially large. F2 will then pick a random number R between 1 and SN_{max}. When asked to authenticate stream $S^{(R)}$, F2 lets $\alpha^{(R)} = \alpha$ and then follows the CSA authentication algorithm, but choses $k = |S^{(R)}|$. When F stops, it will output S' such that $|S'| > |S'^{(i)}|$, with probability greater than $\epsilon/6$, and $i = R$ with probability $1/SN_{max}$. Hence, with probability greater than $\epsilon/(6 * SN_{max})$, $S'_{|S'^{(i)}|+1} = (S, MAC, h^{k-(|S'^{(i)}|+1)}(\alpha)) = (S, MAC, h^{k-(k+1)}(\alpha)) = (S, MAC, h^{-1}(\alpha))$. Since $h(x) = g_x(0)$, inverting h means breaking g_{key} after observing $g_{key}(0)$.

Case 2.3 $|S'| = |S'^{(i)}| = j$. There are two cases, and one must hold with probability at least $\epsilon/12$:

Case 2.3.1 $S_j = S^{(i)}_j$. This case does not contraddict the Lemma.

Case 2.3.2 $S_j \neq S^{(i)}_j$. Let $S'_j = S_j, MAC_{key}(S_j), h^{k-j+1}(\alpha)$. There are two cases, and at least one must occur with probability at least $\epsilon/24$:

Case 2.3.2.1 $MAC_{key}(S_j)$, generated by F, and $MAC(S^{(i)}_j)$, given in stream $S'^{(i)}$, are valid under the same key. In this case we can use F to break g_{uk}, for some unknown key uk, in a polynomial algorithm F3, that has access to an oracle for g_{uk}. Define again SN_{max} as the maximum number of authenticated streams that F will ask for. F3 will then pick a random number R between 1 and SN_{max}. F3 will run F as a subroutine, will use GA to generate a key pair (SK, PK), and will authenticate all streams requested by F normally using CSA, except for stream $S^{(R)}$. For this stream, define $l = |S^{(R)}|$, and let $\alpha^{(R)} = uk$, and $k = l$. Then, $h^{k-l+1}(\alpha^{(R)}) = h(uk)$. F3 then computes $h^{l-1}(uk), ..., h(uk)$, after querying the oracle for $h(uk) = g_{uk}(0)$, and uses these values, in this order, to compute the MACs for $S^{(R)}_1, ..., S^{(R)}_{l-1}$, as required in CSA. As the last authenticated block, F3 outputs $S'^{(R)}_l = (S^{(R)}_l, MAC_{uk}(S^{(R)}_l), h(uk))$, where $MAC_{uk}(S^{(R)}_l) = g_{uk}(1, S^{(R)}_l)$, is queried to the oracle. With probability $1/SN_{max}$, the authenticated stream S'

output by F is associated to stream $S^{(R)}$, i.e., $i = R$. In this case, $MAC_{uk}(S_j^{(R)})$ and $MAC_{key}(S_j)$, are valid under the same key, i.e., $key = uk$. F3 then outputs $MAC_{uk}(S_j) = g_{uk}(1, S_j)$, and since $S_j \neq S_j^{(i)}$, this is a new forged MAC, and a correct value of g_{uk} for the new input $(1, S_j)$, that is generated with non-negligible probability greater than $\epsilon/(24 * SN_{max})$.

Case 2.3.2.2 $MAC_{key}(S_j)$, generated by F, and $MAC(S_j^{(i)})$, given in stream $S'^{(i)}$, are not valid under the same key. We show that, in this case, there is no polynomial time algorithm F' that can generate with non-negligible probability a valid authenticated continuation $T' \supset S'$, given $S'^{(1)}, ..., S'^{(k)}$, possibly other adaptively chosen authenticated streams $T'^{(1)}, ..., T'^{(p)}$, and any valid continuation of $S'^{(i)}$. Suppose such a forger F' exists, and does the above with non-negligible probability ϵ'. Let $T'_{j+1} = (T_{j+1}, MAC, key)$. Since T' is valid and is a continuation of S', we also know that $T'_j = S'_j = (S_j, MAC_{key}(S_j), h^{k-j+1}(\alpha))$. We construct F4 that can generate collisions for h with non-negligible probability, using F and F'. F4 starts by generating a key pair (SK, PK). Then, it runs F as a subroutine and authenticates the requested streams normally using CSA. F will then output S' satisfying the conditions of this case. We also know that a continuation forger F' exists and therefore S' has a possible valid continuation. Consequently $MAC_{key}(S_j)$ is a valid MAC for some value of key, and for the conditions in this case, $key \neq h^{k-j}(\alpha)$. This all happens with probability greater than $\epsilon/24$. In order to obtain key, and thus obtain a collision for h, F4 must then run F' as a subroutine. F4 will authenticate additional streams asked by F' normally, using CSA, and will then produce a continuation of $S'^{(i)}$, as requested by F', also using CSA. When F' outputs a valid continuation T' of S', this must include the value of key, and F4 outputs $(key, h^{k-j}(\alpha))$ as a collision for h. This must occur with non-negligible probability (greater than $\epsilon * \epsilon'/24$). □

MAC and h are of the CSA algorithm, and are defined through a pseudorandom function [Goldreich et al. 1986; Bellare et al. 1996] as defined in (3) above, because not only should the MAC be secure, but each key k must look random even though $h(k)$ is known. With the definition of (3), knowing $h(k) = g_k(0)$ gives no additional information, as one could in any case query the oracle for g_k and obtain $g_k(0)$. In practice, one could use g = HMAC [Krawczyk et al. 1997] so as to satisfy both (2) and (4).

4. INTERACTIVE CSA (I-CSA)

We will now use the CSA scheme to authenticate information over an insecure network, as initially presented in [Bergadano et al. 2000a]. In fact, CSA's security against continuations can be used with a synchronization mechanism to obtain a very efficient individual authentication method.

4.1 The Chained Stream Authentication Protocol with one Sender and one Receiver

For now, we consider one party, named A, who will send authenticated data, and one party, named B, who will receive the data. The protocol is defined below, where $Sig(SK_A, x, y)$ is A's signature under (G, S, V), and similarly $Sig(SK_B, x, y)$ for B:

1. B $\rightarrow$ A: $h^k(\beta), SN, Sig(SK_B, h^k(\beta), SN)$
   A $\rightarrow$ B: $A_1, MAC_{h^{k-1}(\alpha)}(A_1), h^k(\alpha), SN, Sig(SK_A, h^k(\alpha), SN)$
2. B $\rightarrow$ A: $h^{k-1}(\beta)$
   A $\rightarrow$ B: $A_2, MAC_{h^{k-2}(\alpha)}(A_2), h^{k-1}(\alpha)$

...

i. B $\rightarrow$ A: $h^{k-i+1}(\beta)$
   A $\rightarrow$ B: $A_i, MAC_{h^{k-i}(\alpha)}(A_i), h^{k-i+1}(\alpha)$

...

Messages are sequential: A will not send message i if it has not received a correct i-th message from B, and B will not send message $i+1$ if it has not received from A a correct i-th message. A and B initially generate individual random secrets α and β, and compute $h^k(\alpha)$ and $h^k(\beta)$, respectively. These values, and the session number SN, are signed, and exchanged as part of the first messages in step 1. To avoid a possible man-in-the-middle attack, the session number must also contain the identifiers of the entities participating in the protocol. An explanation for this constraint will be given in section 4.3. Then, A sends data as defined in the CSA scheme, and B sends back authenticated acknowledgments. B's authenticated ack for A's j-th message is simply $h^{k-j}(\beta)$. The receiver side is then similar to what happens in S/Key and similar applications [Haller 1994; Lamport 1981]. The security properties of the protocol, to be discussed next, are made relative to the following:

Definition 8. [Active Attack Model]: the **Active Attack Model** with CSA sender A, CSA receiver B, and attacker E is the following:

—E runs in polynomial time and may ask A to send B the authenticated streams $S^{(1)}, ..., S^{(k)}$ in sessions $1, ..., k$.

—A chooses stream $S^{(k+1)}$ and sends it to B in session $k+1$.

—At any time, E can read messages, stop messages and insert messages.

—During session $k+1$, E tries to have B receive $S' \neq S^{(k+1)}$ and believe it authentic.

We call the above an *active stream authentication attack*. We shall prove in Theorem 1 that such attacks are not feasible with the above CSA protocol, except for the possible falsification of the last block of $S^{(k+1)}$. We first note that session numberings by sender and receiver are consistent:

Observation 1. Suppose A has sent its first message of session SN. Then B must have already sent its first message of session SN.

PROOF. (of Observation 1.) We construct an algorithm F that simulates A and B over an insecure network, where E can perform active attacks. F controls the simulations of A and B out of band, i.e. over a secure, separate channel. Suppose that, with non-negligible probability, E has taken action so that B has not yet sent the first message of session SN, when A has already sent its first message of session SN. Then we construct F so that it can forge signatures under the asymmetric scheme (G, S, V). F simulates A normally, by first calling GA to generate a key pair (SK_A, PK_A). For simulating B, F does not generate a key pair, but relies on the oracle for the signature required in the first message of every session. At some

point, F's simulation of A must have computed the first message of session SN, and it must therefore have received $Sig(SK_B, H, SN)$. However, we have supposed F's simulation of B has not yet computed the first message of session SN. As a consequence, $Sig(SK_B, H, SN)$ was never queried to the oracle, and can be output as a forged signature. □

Observation 2. Suppose B has sent its second message of session SN. Then A must have already sent its first message of session SN.

PROOF. (of Observation 2.) Under the same setting of Observation 1, F simulates B normally, by first calling GA to generate a key pair (SK_B, PK_B). For simulating A, F does not generate a key pair, but relies on the oracle for the signature required in the first message of every session. At some point F's simulation of B must have sent the second message of session SN, and therefore it must have received $Sig(SK_A, H, SN)$. However, since F's simulation of A has not yet begun running session SN, it has not yet computed the first authenticated block. As a consequence, $Sig(SK_A, H, SN)$ was never queried to the oracle, and can be output as a forged signature. □

The observations allow us to speak of a "current session" in the CSA protocol with one sender and one receiver. We can now prove that this protocol represents a valid authentication mechanism:

THEOREM 1. ***(Security of CSA with one sender and one receiver):*** *Suppose sender A and receiver B run SN sessions under the CSA protocol, where:*

—*the conditions of Lemma 1 hold;*
—*a polynomial active attacker E chooses streams* $A^{(1)}, ..., A^{(SN-1)}$, *that A sends to B in sessions* $1, ..., SN-1$;
—*B has received the valid authenticated stream* $S' = S'_1, ..., S'_j$ *during session SN.*

Then, E can cause $S'_1, ..., S'_{j-1}$ *to be non-authentic only with negligible probability.*

We shall now prove Theorem 1 by induction on $|S'|$, based on Lemma 1. However, we first need the following, that characterizes the information available to an active attacker at any given moment:

LEMMA 2. *Suppose A and B run session SN, where:*
(1) (G, S, V) is a secure signature scheme, and
(2) h is a one-way hash function
Suppose also that B has received, during session SN, the valid authenticated stream $S' = S'_1, ..., S'_{j-1}$, *and no more. Then, there is no active attacker E that can, with non-negligible probability, cause A to release more than j authenticated blocks during session SN.*

PROOF. (of Lemma 2.)
Let parties A and B run session SN of the CSA protocol with one sender and one receiver. Suppose the Lemma is false. Then there is an active attacker E that can, with non-negligible probability, cause a situation where A has released the stream $A' = A'_1, ..., A'_{j+1}$, while B has only received $j-1$ blocks $S'_1, ..., S'_{j-1}$. Since A has released $j+1$ blocks, it must have received authenticated acknowledgements

$h^k(\beta), ..., h^{k-j}(\beta)$. There are two cases, and one must hold with probability at least $\epsilon/2$:

Case 1: $\beta \neq \beta^{(SN)}$, the secret value chosen by B for session SN. Then, we show that we can construct an algorithm F1 that can simulate A and B, and use a signature oracle to forge signatures under (G, S, V). F1 simulates A by running the sender side of the CSA protocol. F1 simulates B by running the receiver side of the CSA, but uses the signature oracle to produce the signatures needed in the first authenticated block of each session. When E has caused the situation covered by this case, F1's simulation of A must have received $Sig(SK, \beta, SN)$, and since $\beta \neq \beta^{(SN)}$, this can be output as a new forged signature.

Case 2: $\beta = \beta^{(SN)}$. Then, we can construct F2 that can compute $h^{-1}(x)$, for any x, with non-negligible probability. F2 will simulate A and B, running the CSA protocol over a network where E can perform active attacks. F2 simulates A by running the sender side of the CSA protocol. F2 simulates B by running the receiver side of the CSA, but first sets the following values:
- pick SN at random between 1 and SN_{max}, where SN_{max} is the maximum number of sessions that the attacker is able to cover;
- pick j at random between 1 and j_{max}, where j_{max} is the maximum number of blocks per session in E's active attacks;
- let $k = j - 1$ and $\beta^{(SN)} = h^j(x)$;

B is then able to send to A all acknowledgments required for receiving the first $j - 1$ blocks, i.e., $h^k(\beta^{(SN)}) = h^{2j-1}(x), ..., h^{k-(j-1)}(\beta^{(SN)}) = x$. When E has caused the situation covered by this case, F2's simulation of A must have received $h^{k-j}(\beta^{(SN)}) = h^{-1}(x)$. This happens with non-negligible probability $\epsilon/2j_{max}SN_{max}$. □

PROOF. (of Theorem 1.)

We prove a stronger claim by induction on $|S'| = j$, namely: under the conditions of the Theorem, the thesis holds and, if S'_j is non-authentic, then E can cause B to receive a valid continuation of S' only with negligible probability.

Base: We show that the claim is true for $j = 1$. Before B has received from A the first message of session SN, by Lemma 2, A has released no more than the first block of session SN. Therefore, the information available to E consists of:
- the previous authenticated streams $A'^{(1)}, ..., A'^{(SN-1)}$ sent by A, and
- the first block $A_1'^{(SN)}$ of session SN.

Let $S'_1 = S_1, MAC_1, H, SN, Sig(SK, H, SN)$. Since S' is valid, by Lemma 1, there is $q \in [1, SN]$ such that $S_1'^{(q)} = S_1^{(q)}, MAC_1^{(q)}, H, SN, Sig(SK, H, SN)$. Since sequence number SN is only used in $A'^{(SN)}$, clearly the only value for q is SN. Also by Lemma 1, if $S'_1 \neq A_1'^{(SN)}$, then there is no polynomial algorithm that can generate a valid authenticated continuation of S', given adaptively chosen streams, and any valid continuation of A'. So neither can E, even after obtaining $A_2'^{(SN)}$.

Inductive step: Suppose that the inductive claim is true for $j - 1$. We prove that it is also true for j. In order to do so, suppose B has received the valid stream $S' = (S'_1, ..., S'_j)$. This means that it was possible to produce a valid continuation of $(S'_1, ..., S'_{j-1})$, and, by the inductive hypothesis, $S_1, ..., S_{j-1}$ must be authentic with probability $1 - \eta$, where η is negligible. We now have to prove that, if S_j is non-authentic, then E can cause B to receive a valid continuation of S' only with negligible probability. By Lemma 2, before B's receipt of S'_j, A has released at

most j blocks during session SN. Therefore, the information available to E before B's receipt of S'_j consists of:
- the previous authenticated streams $A'^{(1)}, ..., A'^{(SN-1)}$ sent by A, and
- the stream $A'^{(SN)} = A'^{(SN)}_1, ..., A'^{(SN)}_j$ sent by A during session SN.

Let $S'_1 = S_1, MAC_1, H, SN, Sig(SK, H, SN)$. Since S' is valid, by Lemma 1, there is $q \in [1, SN]$ such that $S'^{(q)}_1 = S^{(q)}_1, MAC^{(q)}_1, H, SN, Sig(SK, H, SN)$. Since sequence number SN is only used in $A'^{(SN)}$, clearly the only value for q is SN. Also by Lemma 1, if $S'_j \neq A'^{(SN)}_j$, then there is no polynomial algorithm that can generate a valid authenticated continuation of S', given adaptively chosen streams, and any valid continuation of $A'^{(SN)}$. So neither can E, even after obtaining $A'^{(SN)}_{j+1}$. □

The last block of S' may be modified by E. Thus, authentication is obtained with a one block delay: the receiver can ascertain the origin and the integrity of a block in the stream only after receiving the next block. This is acceptable in most multicast applications. This is achieved without shared secrets and without signatures after the first message. The important consequences of this fact are discussed in the next session, where the protocol is used with more than just two parties.

4.2 The N-party I-CSA Protocol

We will now extend the protocol so that it can be used effectively in a multicast conferencing scenario. As a first step, we will consider two parties, A and B, that must exchange authenticated data in both directions. Obviously, this can be done by simply applying the CSA protocol with sender A and receiver B, and simultaneously with sender B and receiver A. However, we can make this more efficient by merging the hash values sent as acknowledgments and the ones sent as hashes of secrets. This results in the following **two-party protocol**:

1. B → A: $B_1, MAC_{h^{k-1}(\beta)}(B_1), h^k(\beta), SN, Sig(SK_B, h^k(\beta), SN)$
   A → B: $A_1, MAC_{h^{k-1}(\alpha)}(A_1), h^k(\alpha), SN, Sig(SK_A, h^k(\alpha), SN)$
2. B → A: $B_2, MAC_{h^{k-2}(\beta)}(B_2), h^{k-1}(\beta)$
   A → B: $A_2, MAC_{h^{k-2}(\alpha)}(A_2), h^{k-1}(\alpha)$

...

i. B → A: $B_i, MAC_{h^{k-i}(\beta)}(B_i), h^{k-i+1}(\beta)$
   A → B: $A_i, MAC_{h^{k-i}(\alpha)}(A_i), h^{k-i+1}(\alpha)$

...

The above protocol may cause practical transmission difficulties. In particular, each party may send a block of data only after receiving a corresponding block from the other party. This causes a kind of stop-and-wait behaviour that implies poor network utilization and may result in unacceptable delays for real-time traffic. Fortunately, such strict sequentialization of messages is not necessary. In particular, data blocks and MACs can be sent at any time - only the delivery of keys need be delayed until acknowledgements are obtained and verified. We may therefore rewrite the above two party protocol by splitting the behaviour of each party into a *data sender* process and a *key sender* process, that run independently. For party A, the processes are defined as follows (party B is defined symmetrically):

A's data sender process:

1. send to B: $MAC_{h^{k-1}(\alpha)}(A_1)$, A_1
2. send to B: $MAC_{h^{k-2}(\alpha)}(A_2)$, A_2

...

A's key sender process:

1. wait for $MAC_{h^{k-1}(\beta)}(B_1)$
   send to B: $h^k(\alpha), SN, Sig(SK_A, h^k(\alpha), SN)$
2. wait for $MAC_{h^{k-2}(\beta)}(B_2)$
   wait for $h^k(\beta), SN, Sig(SK_B, h^k(\beta), SN)$
   send to B: $h^{k-1}(\alpha)$
3. wait for $MAC_{h^{k-3}(\beta)}(B_3)$
   wait for $h^{k-1}(\beta)$
   send to B: $h^{k-2}(\alpha)$

...

Delaying just secrets, not information, is essential in multicast conferencing: the receiver will continue viewing the stream of data even though the keys necessary to authenticate it are not yet available. When secrets are late, viewing is ahead of authentication, and we call this an *authentication delay*. The delay would be small and roughly equivalent to three times the network latency. The reason is that a MAC must be sent, then the authenticated acknowledgement is returned, and finally the MAC key is sent. Only then can the corresponding block be authenticated by the receiver.

We may now generalize the above construction and obtain the **N-party protocol**. During session SN, party i first generates a random secret α_i and computes $h^k(\alpha_i)$. Party i consists of two processes, a *data sender* and a *key sender*, that run concurrently as described below, where $A_{i,j}$ is the jth block sent by party i:

Data sender i:

1. multicast $MAC_{h^{k-1}(\alpha_i)}(A_{i,1})$ and $A_{i,1}$;
2. multicast $MAC_{h^{k-2}(\alpha_i)}(A_{i,2})$ and $A_{i,2}$;

...

Key sender i:

1. wait for $MAC_{h^{k-1}(\alpha_j)}(A_{j,1})$, for all $j \in [1, N]$;
   multicast $h^k(\alpha_i), SN, i, Sig(SK_i, h^k(\alpha_i), SN, i)$;
2. wait for $MAC_{h^{k-2}(\alpha_j)}(A_{j,2})$, for all $j \in [1, N]$;
   wait for $h^k(\alpha_j), SN, j, Sig(SK_j, h^k(\alpha_j), SN, j)$, for all $j \in [1, N]$;
   multicast $h^{k-1}(\alpha_i)$;
3. wait for $MAC_{h^{k-3}(\alpha_j)}(A_{j,3})$, for all $j \in [1, N]$;
   wait for $h^{k-1}(\alpha_j)$, for all $j \in [1, N]$;
   multicast $h^{k-2}(\alpha_i)$;

...

It is important to note that, for every block, the only authentication information that is multicast is one MAC (sent by the data sender) and one hash value (sent by the key sender). This does not grow with N.

4.3 Considerations on a possible man-in-the-middle attack

It is worth to note that indicating explicitly the name of sender(s) and receiver(s) in the SN, our protocol does not suffer from man-in-the-middle attacks from outsiders the group and also from members of the group. For simplicity we show this attack on a communication of two parties, A and B; the solution to the attack can be generalized to the N-party protocol, even if the attacker I, that impersonates A, $I(A)$, is a member of the group.

Let us see the attack:

1. B $\rightarrow$ I(A): $B_1, MAC_{h^{k-1}(\beta)}(B_1), h^k(\beta), SN, Sig(SK_B, h^k(\beta), SN)$

1'. I $\rightarrow$ I(A): $I_1, MAC_{h^{k-1}(\gamma)}(I_1), h^k(\gamma), SN, Sig(SK_I, h^k(\gamma), SN)$
A $\rightarrow$ I: $A_1, MAC_{h^{k-1}(\alpha)}(A_1), h^k(\alpha), SN, Sig(SK_A, h^k(\alpha), SN)$

2'. I $\rightarrow$ A: $I_2, MAC_{h^{k-2}(\gamma)}(I_2), h^{k-1}(\gamma)$
A $\rightarrow$ I: $A_2, MAC_{h^{k-2}(\alpha)}(A_2), h^{k-1}(\alpha)$

2. I(A) $\rightarrow$ B: $A'_1, MAC_{h^{k-1}(\alpha)}(A'_1), h^k(\alpha), SN, Sig(SK_A, h^k(\alpha), SN)$

and from now on, I can forge all A's messages.

This attack is avoided if the session number SN also contains the identifiers of the participants in the communication; these identifiers are also needed by the participants because the protocol requires to wait the signatures, MACs and keys from all the participants. And in this case, A would not accept the message from I because it is signed by I but the session is for A and B. In the multiparty case, an intruder external to the group cannot forge any message for the reason above. Instead, a participant of the group cannot masquerade as another participant because the shown attack requires that A releases two keys in the beginning, but A will not release the second key if she has not received the signatures and all the previous MACs and keys from the other participants. In this situation, forging cannot be performed by anyone.

5. TIMED CSA (T-CSA)

In this section we apply the CSA scheme to network applications where receivers do not transmit data and are unable to send acknowledgements in due time. This section is based on our initial presentation in [Bergadano et al. 2000b]. This protocol works with one-directional transmission from one sender to any number of receivers (broadcast transmission). Given that there is no feedback from the receivers, the security of this protocol is based on the time of transmission of the authentication information.

The technique we propose here is especially suited for efficient data authentication in broadcast, or multicast with large groups. It is based on hash chains, but does not suffer from the delay-related problems of the Guy Fawkes protocol. The authentication scheme is detailed below, separately for the broadcast sender,

and for the receivers. The sender sends data A_i and authentication information in separate streams (corresponding to the Data Sender and Key Sender procedures defined below). The sender also announces the session, including some essential security information. Receivers consume data and verify authenticity in two separate processes (Receiver and Authenticator).

5.1 Sender

Announcement:
broadcast the following:
session number SN,
starting time ST,
$\alpha_k = h^k(\alpha)$ (where α is a random secret,
h is a collision-resistant hash function),
maximum time imprecision $\epsilon_s \geq 0$ at sender;
signature $Sig(SK_A, SN, ST, \alpha_k, \epsilon_s)$

Data sender:
broadcast $MAC_{h^{k-1}(\alpha)}(A_1), A_1$ (at time ST)
broadcast $MAC_{h^{k-2}(\alpha)}(A_2), A_2$ (at time ST+T)
...

Key sender:
wait until time ST+delay, broadcast $\alpha_{k-1} = h^{k-1}(\alpha)$
wait until time ST+delay+T, broadcast $\alpha_{k-2} = h^{k-2}(\alpha)$
wait until time ST+delay+2T, broadcast $\alpha_{k-3} = h^{k-3}(\alpha)$
...

The delay is needed so that keys are not made public too early, i.e., before corresponding data and MACs are received. A good choice could be
delay=latency+$\epsilon_s + max_{r \in Receivers}(\epsilon_r) + R$,
where $\epsilon_r \geq 0$ is the receiver clock precision, i.e. the maximum time-advance imprecision of the receivers' clocks, R is a reliability parameter, and latency is the estimated maximum propagation time. T is the time length interval of each data packet; in case of audio transmission, T is the time length of sampled audio that is contained in one packet.

5.2 Receiver

Receiver:
repeat forever:
receive X (where X is a MAC, a data block,
or a key);
mark receipt time;
store X;
if X is a data block, consume X;

Authenticator:
receive session announcement

$SN, ST, \alpha_k, \epsilon_s, Sig(SK_A, SN, ST, \alpha_k, \epsilon_s)$
if $Sig(SK_A, SN, ST, \alpha_k, \epsilon_s)$ is wrong, exit;
for $i = 1$ to k do :
    wait until all of the following is received:
        $MAC_{\alpha_{k-i}}(A_i)$,
        A_i,
        α_{k-i};
    if $h(\alpha_{k-i}) \neq \alpha_{k-i+1}$ then exit,
    if $MAC_{\alpha_{k-i}}(A_i)$ is *late*,
        mark authentication of A_i as **unknown**;
    if $MAC_{\alpha_{k-i}}(A_i)$ is wrong,
        mark A_i as **forged**;
    else mark A_i as **authentic**;

We must define the meaning of "late" in the authenticator procedure. Informally, the MAC is late when received after the corresponding key was released. However, we must consider that sender time and receiver time differ, hence:

$MAC_{\alpha_{k-i}}(A_i)$ is late when received at time $TM > ST + (i-1) * T - \epsilon_s - \epsilon_r + \text{delay}$.

In fact, if the MAC is received after this deadline, then the key could have been released earlier, and with that key anybody could have generated the MAC. In this case, the block is received and consumed, but is marked as possibly non-authentic. Further blocks may again be authenticated normally, hence abnormal network delays cause authentication "holes", but no other security problem. The choice of the key sending delay time is important: large values of the delay will reduce the number of authentication holes, but will cause authentication decisions to be delayed with respect to data consumption. Small values of delay allow the receivers to authenticate data shortly after it has been played, but lead to a risk of blocks with "unknown" authenticity.

This protocol is robust to network losses, in fact if a key is lost, only the packet authenticated with that key will have unknown authentication, while the other packets will be verified normally, because the hash function can be applied as many times as required to produce an already autheticated key (in the extreme case, to arrive to the signed key). Moreover, if the key is authentic, all the generated (but previously lost) keys may be used as if they were correctly received to authenticate the corresponding data packets (see Figure 5.2). In fact, suppose that key α_{i+1} is authentic and that the key α_i is lost. If key α_{i-1} is correctly received, it can be verified with two applications of h, i.e. α_{i-1} is authentic if and only if $h(h(\alpha_{i-1})) = \alpha_{i+1}$, and if this test is true, from $h(\alpha_{i-1})$ we also generate α_i that is authentic by definition and may be used to verify the corresponding packet. If a MAC is lost, only the corresponding packet will be affected with an unknown authentication.

6. THE MULTICAST AUTHENTICATION TOOL

We have developed an authentication tool called *Multicast Authentication Tool* (MAT) that implements the CSA with time synchronization (T-CSA) presented

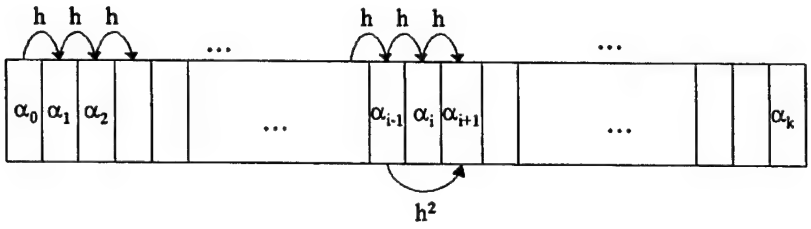


Fig. 1. The hash chain of keys.

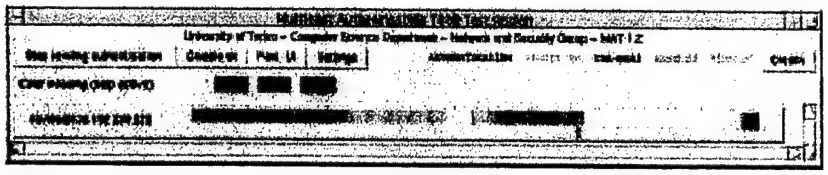


Fig. 2. The user interface of the MAT authentication application.

in the previous section. MAT has been designed as a separate module that we integrated into the code of the Robust Audio Tool [Robust Audio Tool]. The module can however be easily integrated in any audio/video application that requires individual authentication. RAT is a well known audio tool developed by the University College of London Networked Multimedia Research Group with the goal of providing an application that allows users to participate in multicast conferences over the MBone. MAT uses parts of the data structures of RAT and adds its own. The MAT functions are added to the functions of RAT and are called in various parts of the RAT code. Ideally, we have an engine that integrates into the engine of RAT and a new user interface that communicates with the engine. The user interface is shown in Figure 6. In our implementation we have a more complex user interface, that allows to control the reception of the various packets of the protocol, allowing experimenting with packet losses, packet delaying and packet modification.

The Robust Audio Tool sends the audio and control packets according to the RTP/RTCP [Schulzrinne et al. 1996] protocol for real time data transmission. The RTP/RTCP traffic is sent on two consecutive transport ports. We use a new transport port, the *security port* next after the RTCP port, for traffic relative to the authentication protocol: in this way we do not interfere with the RTP/RTCP traffic of RAT. The MACs could be inserted in the extension part of the RTP packet but given that not all the audio applications deal with the extensions (they simply discard the packet if there are extensions) we also sent the MAC as separate data on the security port.

MAT allows two types of authentication: RSA, where each audio packet has an associated signature packet on the security port (with no time constraints) and T-CSA.

All the packets of our authentication protocol flow on the channel defined by the security port. The possible packets related to the T-CSA protocol are MAC packets,

where the MAC are obtained from the chained keys of the CSA, signed keys that sign the announcement and define session numbers and the last key of the chain, and key packets, each one containing a key of the chain. When the application runs the RSA based protocol, and signs each audio packet, also the signature packets are multicast on the security port. Moreover, when the application is started, the certificate of the participant is sent on the security port.

The T-CSA implementation works in phases: each phase consumes k keys. In each phase a signature of $[SN, ST, h^k(\alpha), \epsilon_s]$ is released, along with the chain of keys. Each key is released at the correct instant of time according to the protocol in Section 5.1. Meanwhile, the audio packets and the corresponding MACs are sent as soon as possible. In this way we have an asymmetric signature and k MAC calculations: our implementation uses RSA for the signature and HMAC [Krawczyk et al. 1997], with SHA-1, for the MAC.

Given that the keys are consumed from the beginning of the transmission of the session, a key generation process builds a new chain in such a way that when the last key of the old chain is consumed, the first one of the new chain is available.

6.1 Performance

The advantage of the CSA protocol is that it uses a very small authentication information (a MAC and a key, of the same size of a MAC, per packet) and that this information is very fast to compute with respect to an RSA signature, in fact CSA, after the first message, requires to compute just two hash values to authenticate each message. RSA could be made more efficient by simply using a single RSA signature to authenticate many packets. Of course the drawbacks of this solution are that in case of packet loss, though, all the packets under the same signature will result in an "unknown" authentication. Moreover, having the authentication extend onto many packets leads to a delayed knowledge of the authentication status of the received packets: for example, authenticating five minutes of audio data with just one signature, means that receivers know that what they heard is authentic only at the end of the five minutes. This has as the consequence that they cannot trust what they heard until the end of the five minutes.

Actually, the T-CSA protocol may be also implemented in multicast video conferencing tools.

7. CONCLUSIONS AND THE NON-REPUDIATION ISSUE

The problem of individual authentication in multicast applications poses novel security problems. Proposed solutions have to deal with efficiency and scalability issues, and should adapt to the evolution of protocols and the network and transport levels.

In this paper we have shown two possible solutions to the problem of individual authentication in the group communication context. Both are based on the concept of hash chain, linking the authentication information of each data packet. These solutions are very efficient because they are based on MACs, and use an asymmetric digital signature only once, to bootstrap and to authenticate the hash chain. Moreover, the amount of authentication information does not depend on the number of receivers that have to authenticate the data.

The first proposed solution, I-CSA, is an interactive stream authentication pro-

tocol, that is proved to be secure against non-authentic stream continuations.

The second protocol, T-CSA, is based on the same concept of hash chain, but does not require a continuous exchange of data among the parties; instead, it is tailored for multicast/broadcast applications in which the data flows in one direction only (from sender to receivers). Given that there is no feedback from the receivers, the security of this protocol is based on time synchronization among the participants, and on the time when the authentication information is released. Nonetheless, T-CSA may be used when there is continuous exchange of data among the participants.

Due to the use of MACs, the proposed schemes have shown to be very efficient, and the implementation of T-CSA in a widely used audio application has demonstrated that the overhead of this protocol is compatible with the load of a machine performing multimedia processing in the context of a teleconference.

Our last consideration is in order. In a recent paper [Boneh et al. 2000], Boneh et al. prove that any secure multicast MAC whose length is slightly less than the number of receivers can be converted into a signature scheme, hence providing non-repudiation. On the other hand, CSA is secure, its length is independent of the number of receivers, and it is not a signature. We believe that the reason for this apparent contradiction is that the formalization of Boneh et al. concerns the authentication of a message M, not a stream S. CSA does not generate a single authentication code for the whole stream, it creates separate codes for every block, and such codes are released either after receipt acknowledgements (I-CSA) or based on time constraints (T-CSA). Hence one cannot just concatenate the separated codes to form one MAC for the streams.

It is true, though, that we do obtain non-repudiation in front of subjects that participate in the protocol. For T-CSA this means just "listening" to the messages. This feature also belongs the Guy Fawkes protocol [Anderson et al. 1998], that was in fact presented initially as a form of "symmetric signature".

ACKNOWLEDGMENTS

The authors thank Dan Boneh for providing input on his recent work on multicast MACs, Ran Canetti for suggesting the use of pseudorandom functions, Rosario Gennaro for reviewing early versions of the paper, Adrian Perrig for his comments on the interactive protocol, and the people of the RAT project for the insights on the RAT source code.

REFERENCES

ANDERSON, R., BERGADANO, F., CRISPO, B., LEE, J., MANIFAVAS, C., AND NEEDHAM, R. 1998. A new family of authentication protocols. *ACM Operating Systems Review 32*, 4, 9–20.

ASOKAN, N., TSUDIK, G., AND WAIDNER, M. 1996. Server supported signatures. In *Proceedings Esorics '96* (September 1996), pp. 131–143.

BELLARE, M., CANETTI, R., AND KRAWCZYK, H. 1996. Pseudorandom functions revisited: The cascade construction and its concrete security. In *Proc. 37th Symposium on the Foundations of Computer Science* (1996). IEEE.

BERGADANO, F., CAVAGNINO, D., AND CRISPO, B. Individual authentication for multicast conferencing.

BERGADANO, F., CAVAGNINO, D., AND CRISPO, B. 2000a. Chained stream authentication. In *Proceedings of Selected Areas in Cryptography 2000* (August 2000), pp. 142–155.

BERGADANO, F., CAVAGNINO, D., AND CRISPO, B. 2000b. Individual single source authentication on the mbone. In *Proceedings International Conference on Multimedia and Expo 2000* (August 2000), pp. 541–544.

BLEICHENBACHER, D. AND MAURER, U. 1996. On the efficiency of one-time digital signatures. In *Advances in Cryptology - AsiaCrypt '96, LNCS 1163* (1996). Springer-Verlag.

BONEH, D., DURFEE, G., AND FRANKLIN, M. 2000. Lower bounds for multicast message authentication. *Submitted.*

CANETTI, R. AND PINKAS, B. 1998. A taxonomy of multicast security issues. *Internet Draft.*

CHEUNG, S. 1997. An efficient message authentication scheme for link state routing. In *Proc. 13th Annual Computer Security Application Conference* (December 1997).

DYER, M., FENNER, T., FRIEZE, A., AND THOMASON, A. 1995. On key storage in secure networks. *Journal of Cryptology 8*, 189–200.

EVEN, S., GOLDREICH, O., AND MICALI, S. 1989. On-line/off-line digital signatures. In *Advances in Cryptology - Crypto '89, LNCS 435* (1989), pp. 263–275. Springer-Verlag.

GENNARO, R. AND ROHATGI, P. 1997. How to sign digital streams. In *Advances in Cryptology - Crypto '97, LNCS 1294* (1997), pp. 180–197. Springer-Verlag.

GOLDREICH, O., GOLDWASSER, S., AND MICALI, S. 1986. How to construct random functions. *Journal of the ACM 33*, 4, 210–217.

GOLDWASSER, S., MICALI, S., AND RIVEST, R. 1988. A digital signature scheme secure against adaptive chosen message attacks. *SIAM Journal of Computing 17*, 2 (April), 281–308.

HALLER, N. M. 1994. The s/key(tm) one-time password system. In *Proc. 1994 ISOC Symposium on Network and Distributed Security* (February 1994).

KRAWCZYK, H., BELLARE, M., AND CANETTI, R. 1997. Hmac: Keyed-hashing for message authentication. *RFC 2104*.

LAMPORT, L. 1979. Constructing digital signatures from a one-way function.

LAMPORT, L. 1981. Password authentication with insecure communication. *Communication of the ACM 24*, 11 (November), 770–772.

MERKLE, R. C. 1987. A digital signature based on a conventional encrytpion function. In *Advances in Cryptology - Crypto '87, LNCS 293* (1987), pp. 369–378. Springer-Verlag.

PERRIG, A., CANETTI, R., TYGAR, J. D., AND SONG, D. 2000. Efficient authentication and signing of multicast streams over lossy channels. In *Proc. of the IEEE Security and Privacy Symposium* (May 2000).

ROBUST AUDIO TOOL. *Networked Multimedia Research Group.* http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/.

SCHULZRINNE, H., CASNER, S., FREDERICK, R., AND JACOBSON, V. 1996. Rtp: A transport protocol for real-time applications. *RFC 1889*.

WONG, C. K. AND LAM, S. S. 1998. Digital signatures for flows and multicasts. In *Proceedings of IEEE ICNP'98* (October 1998).

ZHANG, K. 1998. Efficient protocols for signing routing messages. In *Proc. 1998 Symposium on Network and Distributed System Security* (March 1998).

How to Sign Digital Streams

Rosario Gennaro * Pankaj Rohatgi *

July 6, 2000

Abstract

We present a new efficient paradigm for signing digital streams. The problem of signing digital streams to prove their authenticity is substantially different from the problem of signing regular messages. Traditional signature schemes are message oriented and require the receiver to process the entire message before being able to authenticate its signature. However, a stream is a potentially very long (or infinite) sequence of bits that the sender sends to the receiver and the receiver is required to consume the received bits at more or less the input rate and without excessive delay. Therefore it is infeasible for the receiver to obtain the entire stream before authenticating and consuming it. Examples of streams include digitized video and audio files, data feeds and applets.

We present two solutions to the problem of authenticating digital streams. The first one is for the case of a finite stream which is entirely known to the sender (say a movie). We use this constraint to devise an extremely efficient solution. The second case is for a (potentially infinite) stream which is not known in advance to the sender (for example a live broadcast). We present proofs of security of our constructions. Our techniques also have applications in other areas, for example, efficient authentication of long files when communication is at a cost and signature-based filtering at a proxy server.

1 Introduction

Digital Signatures (see [10, 22]) are the cryptographic answer to the problem of information authenticity. When a recipient receives digitally signed information and she is able to verify the digital signature then she can be certain that the information she received is exactly the same as what the sender (identified by his public key) has signed. Moreover, this guarantee is *non-repudiable*, i.e., the entity identified by the public key cannot later deny having signed the information. Thus, the recipient can hold the signer responsible for the content she receives. This distinguishes digital signatures from *message authentication codes* (MAC) which allow the receiver to have confidence on the identity of the sender, but not to prove to someone else this fact, i.e. MAC's are repudiable.

However, current digital signature technology was designed to ensure message authentication and its straightforward application does not yield a satisfactory solution when applied to information resources which are not message-like. In this paper we discuss one such type

*I.B.M. T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, U.S.A. Email: {rosario,rohatgi}@watson.ibm.com A preliminary version of this paper appeared in the proceedings of CRYPTO'97.

of resource: *streams*. We point out shortcomings in several approaches (some of them used in practice) to tackle the problem of signing streams and then present our solution which does not have such shortcomings.

1.1 Streams Defined

A stream is a potentially very long (infinite) sequence of bits that a sender sends to a receiver. The stream is usually sent at a rate which is negotiated between the sender and receiver or there may be a demand-response protocol in which the receiver repeatedly sends requests for additional (finite) amount of data. The main feature of streams which distinguish them from messages is that the receiver must consume the data it receives at more or less the input rate, i.e., it can't buffer large amounts of unconsumed data. In fact in many applications the receiver stores relatively very small amounts of the stream. In some cases the sender itself may not store the entire sequence, i.e., it may not store the information it has already sent out and it may not know anything about the stream much beyond of what it has sent out.

There are many examples of digital streams. Common examples include *digitized video* and *audio* which is now routinely transported over the Internet and also to television viewers via various means, e.g., via direct broadcast satellites and very shortly via cable, wireless cable, telephone lines etc. This includes both pre-recorded and stored audio/video programming as well as live feeds. Apart from audio/video, there are also *data feeds* (e.g., news feeds, stock market quotes etc) which are best modeled as a stream. The Internet and the emerging interactive TV industry also provide another example of an information resource which is best modeled as a stream, i.e., *applets*. Most non-trivial applets are actually very large programs which are organized into several modules. The consumer's machine first downloads and executes the startup module and as the program proceeds, additional modules are downloaded and executed. Also, modules that are no longer in use may be discarded by the consumer machine. This structure of applets is forced by two factors. Firstly, the amount of storage available on the consumer machine may be limited (e.g., in the emerging interactive TV industry set-top boxes have to be cheap and therefore resource limited) . Secondly (in the case of the Internet), the bandwidth available to download code may be limited and applets must be designed to start executing as soon as possible. Also it is quite likely that some of the more sophisticated applets may have data-rich components generated on the fly by the applet server. Therefore applets fit very nicely into the demand/response streams paradigm.

Given the above description, it is clear that message-oriented signature schemes cannot be directly used to sign streams since the receiver cannot be expected to receive the entire stream before verifying the signature. If a stream is infinitely long (e.g., the 24-hours news channel), then it is impossible for the receiver to receive the entire stream and even if a stream is finite but long the receiver would have to violate the constraint that the stream needs to be consumed at roughly the input rate and without delay.

1.2 Previous Solutions and their Shortcomings

Up to the authors' knowledge there has been no proposed specific solution to the problem of signing digital streams in the crypto literature. One can envision several possible solutions,

some of them are actually proposed to be used in practice.

One type of solution splits the stream in blocks. The sender signs each individual block and the receiver loads an entire block and verifies its signature before consuming it. This solution also works if the stream is infinite. However, this solution forces the sender to generate a signature for each block of the stream and the receiver to verify a signature for each block. With today's signature schemes either one or both of these operations can be very expensive computationally. Which in turns means that the operations of signing and verifying can create a bottleneck to the transmission rate of the stream.

Another type of solution works only for finite streams which are known in advance. In this case, once again the stream is split into blocks. Instead of signing each block, the sender creates a table listing cryptographic hashes of each of the blocks and signs this table. When the receiver asks for the authenticated stream, the sender first sends the signed table followed by the stream. The receiver first receives and stores this table and verifies the signature on it. If the signature matches then the receiver has the authenticated cryptographic hash of each of blocks in the stream and thus each block can be verified when it arrives. The problem with this solution is that it requires the storage and maintenance of a potentially very large table on the receiver's end. In many realistic scenarios the receiver buffer is very limited compared to the size of the stream, (e.g., in MPEG a typical movie may be 20 GBytes whereas the receiver buffer is only required to be around 250Kbytes). Therefore the hash table can itself become fairly large (e.g., 50000 entries in this case or 800Kbytes for the MD5 hash function) and it may not be possible to store the hash table itself. Also, the hash table itself needs to be transmitted first and if it is too large then there will be a significant delay before the first piece of the stream is received and consumed. To address the problem of large tables one can also come up with a hybrid scheme in which the stream is split in consecutive pieces and each piece is preceded by a small signed table of contents.[1]

The above solution can be further modified by using an authentication tree: the blocks are placed as the leaves of a binary tree and each internal node takes as a value the hash of its children (see [17].) This way the sender needs to sign and send only the root of this tree. However, in order to authenticate each following block the sender has to send the whole authentication path (i.e., the nodes on the path from the root to the block, plus their siblings) to the receiver. This means that if the stream has k blocks, the authentication information associated with each block will be $O(\log k)$.

As we will see briefly our solution eliminates all these shortcomings. The basic idea works for both infinite and finite streams, only one expensive digital signature is ever computed, there are no big tables to store, and the size of the authentication information associated with each block does not depend on the size of the stream.

DIGITAL SIGNATURES VS. SIMPLE AUTHENTICATION. Notice that if the receiver were only interested in establishing the identity of the sender, a solution based on MAC would suffice. Indeed, once the sender and receiver share a secret key, the stream could be authenticated block by block using a MAC computation on it. Since MAC's are usually faster than

[1]This is the case now (Java Developer Kit 1.1) for large signed java applets which are distributed as a collection of Java archives (JAR) where each archive has a signed table of hashes of contents and the archives are loaded in the order given in the HTML page in which the applet is embedded.

signatures to compute and verify, this solution would not incur the computational cost associated with the similar signature-based solution described above.

However, a MAC-based approach would not enjoy the non-repudiation property. We stress that we require such property for our solution. Also in order for this property to be meaningful in the context of streams we need to require that *each* prefix of the stream to be non-repudiable. That is, if the stream is $\mathcal{B} = B_1, B_2, \ldots$ where each B_i is a block, we require that each prefix $\mathcal{B}_i = B_1 \ldots B_i$ be non-repudiable. This rules out a solution in which the sender just attaches a MAC to each block and then signs the whole stream at the end.

This is to prevent the sender from interrupting the transmission of the stream before the non-repudiability property is achieved. Also it is a guarantee for the receiver. Consider indeed the following scenario: the receiver notices that the applets she is downloading are producing damages to her machine. She interrupts the transfer in order to limit the damage, but at the same time she still wants some proof to bring to court that the substream downloaded so far did indeed come from the sender.

In general non-repudiation is crucial when the stream is being sold as an electronic merchandise. With the advent of music and video distribution over the internet, it is clear that such transactions must be protected with mechanisms that allow the resolution of disputes through non-repudiation.

There are other reasons why a solution based on digital signatures could be preferrable to one based on MAC's. Consider for example the case in which content is being broadcasted to a large number of people which changes over time. In this scenarion (even if non-repudiation is not required) to simply sign the content may end up being the simplest and most efficient solution, since it avoids problems of key management among a large number of users.

1.3 Our Solution in a Nutshell

Our solution makes some reasonable/practical assumptions about the nature of the streams being authenticated. First of all we assume that it is possible for the sender to embed authentication information in the stream. This is usually the case, see Section 8 to see how to do this in most real-world situations like MPEG video/audio. We also assume that the receiver has a "small" buffer in which it can first authenticate the received bits before consuming them. Finally we assume that the receiver has processing power or hardware that can compute a small number of fast cryptographic checksums faster than the incoming stream rate while still being able to play the stream in real-time.

The basic idea of our solution is to divide the stream into blocks and embed some authentication information in the stream itself. The authentication information of the i^{th} block will be used to authenticate the $(i+1)^{st}$ block. This way the signer needs to sign just the first block and then the properties of this single signature will "propagate" to the rest of the stream through the authentication information. Of course the key problem is to perform the authentication of the internal blocks fast. We distinguish two cases.

In the first scenario the stream is finite and is known in its entirety to the signer in advance. This is not a very limiting requirement since it covers most of the Internet applications (digital movies, digital sounds, applets). In this case we will show that a *single* hash computation will suffice to authenticate the internal blocks. The idea is to embed in

the current block a hash of the following block (which in turns includes the hash of the following one and so on...)

The second case is for (potentially infinite) streams which are not known in advance to the signer (for example live feeds, like sports event broadcasting and chat rooms). In this case our solution is less optimal as it requires several hash computations to authenticate a block (although depending on the embedding mechanism these hash computations can be amortized over the length of the block). The size of the embedded authentication information is also an issue in this case. The idea here is to use fast 1-time signature schemes (introduced in [15, 16]) to authenticate the internal blocks. So block i will contain a 1-time public key and also the 1-time signature of itself with respect to the key contained in block $i-1$. This signature authenticates not only the stream block but also the 1-time key attached to it.

1.4 Related Work

The "chaining" technique of embedding the hash of the following block in the current block can be seen as a variation of the Merkle-Damgård meta-method to construct hash functions based on a simpler compression function [18, 9]. The novelty here is that we exploit the structure of the construction to allow fast authentication of single blocks in sequential order. A similar idea had been used in the context of time-stamping mechanisms by Haber and Stornetta [14]. It can also be seen as a weak construction of *accumulators* as introduced in [4]. An accumulator for k blocks $B_1, \ldots, B_k$ is a single value ACC that allows a signer to quickly authenticate any of the blocks in any particular order. Accumulators based on the RSA assumption were proposed in [4]. In our case we have a much faster construction based on collision–free hash functions, since we exploit the property that the blocks must be authenticated in a specific order.

The proofs of security of our schemes are somewhat similar to a proof by Damgard in [8] of the general result that combining a secure signature scheme and a collision intractable hash function yields a secure signature scheme.

Mixing "regular" signatures with 1-time signatures, for the purpose of improving efficiency is discussed in [12]. However, in that paper the focus is in making the signing operation of a message M efficient by dividing it in two parts. An off-line part in which the signer signs a 1-time public key with his long-lived secret key even before the messages M is known. Then when M has to be sent the signer computes a 1-time signature of M with the authenticated 1-time public key and sends out M tagged with the 1-time public key and the two signatures. Notice that the receiver must compute two signature verifications: one on the long-lived key and one on the 1-time key. In our scheme we need to make both signing and verification extremely fast, and indeed in our case each block (except for the first) is signed (and hence verified) only once with a 1-time key.

We also use the idea of using old keys in order to authenticate new keys. This has appeared in several places but always for long-lived keys. Examples include [2, 20, 24] where this technique is used to build provably secure signature schemes. We stress that the results in [2, 20, 24] are mostly of theoretical interest and do not yield practical schemes. Our on-line solution somehow mixes these two ideas in a novel way, by using the chaining technique with 1-time keys, embedding the keys inside the stream flow so that old keys can

authenticate at the same time *both* the new keys and the current stream block.

2 Recent Developments

Subsequent to the publication of a preliminary version of this work in CRYPTO '97, there has been a lot of research activity in the area of source authentication techniques for multicast communications, some of which is applicable to the problem of stream signing. Authenticating the sender of data packets is a fundamental security issue in multicast, and to date, no completely satisfactory solution is known for this problem. Techniques such as the use of MACs, which work well for unicast settings, are completely insecure in multiple receiver settings and much research has focussed on the design of efficient signature schemes for multicast packet authentication. At first glance, it appears that the stream signing work presented in this paper should be applicable to this problem. However, it turns out that on the Internet, most multicast data travels over an unreliable transport and thus the techniques developed in this paper are not directly applicable since they assume reliability. On the other hand, some of the efficient signature mechanisms developed for multicast packet authentication [26, 23] could be used in conjunction with the techniques presented in this paper to yield even better stream signing techniques.

3 Preliminaries

In the following we denote with n the security parameter. We say that a function $\epsilon(n)$ is *negligible* if for all c, there exists an n_0 such that, for all $n > n_0$, $\epsilon(n) < 1/n^c$.

COLLISION-RESISTANT HASH FUNCTIONS. Let $\mathcal{H}$ be a family of functions that map arbitrarily long binary strings into binary strings of a fixed length k. We say that $\mathcal{H}$ is a *collision-resistant* family of hash functions if any polynomial time algorithm who is given as input a description of a random element $H \in \mathcal{H}$, finds a collision, i.e., a pair (x, y) such that $x \neq y$ and $H(x) = H(y)$, only with negligible probability $\epsilon(k)$.

MD5 [21] and SHA-1 [19] are conjectured collision-resistant hash functions, i.e., random representatives of a family $\mathcal{H}$ with the above property.

SIGNATURE SCHEMES. A *signature scheme* is a triplet (G, S, V) of probabilistic polynomial-time algorithms satisfying the following properties:

- G is the *key generation* algorithm. On input 1^n it outputs a pair $(SK, PK) \in \{0,1\}^{2n}$. SK is called the secret (signing) key and PK is called the public (verification) key.
- S is the *signing* algorithm. On input a message M and the secret key SK, it outputs a signature σ.
- V is the *verification* algorithm. For every $(PK, SK) = G(1^n)$ and $\sigma = S(SK, M)$, it holds that $V(PK, \sigma, M) = 1$.

In [13] security for signature schemes is defined in several variants. The strongest variant is called "existential unforgeability against adaptively chosen message attack". That is, we

require that no efficient algorithm will be able to produce a valid signed message, even after seeing several signed messages of its choice.

ONE-TIME SIGNATURES. A special kind of signature schemes satisfy the [13] definition of security only if we allow the adversary to see a limited number of signed messages. In particular there exists signature schemes that are secure only if used to sign a *single* message. The main advantage of this type of schemes is that they are usually much faster to execute than regular signature schemes.

STREAM SIGNATURES. We define a *stream* to be a (possibly infinite) sequence of *blocks* $\mathcal{B} = B_1, B_2, \ldots$ where each $B_i \in \{0,1\}^c$ for some constant[2] c.

We distinguish two cases. In the first case we assume that the stream is finite and known to the sender in advance. We call this the *off-line* case. Conversely in the *on-line* case the signer must process one (or a few) block at the time with no knowledge of the future part of the stream.

Definition 1 *An* off-line stream signature scheme *is a triplet* (G, S, V) *of probabilistic polynomial-time algorithms satisfying the following properties:*

- G *is the* key generation *algorithm. On input* 1^n *it outputs a pair* $(SK, PK) \in \{0,1\}^{2n}$. SK *is called the secret (signing) key and* PK *is called the public (verification) key.*
- S *is the* signing *algorithm. On input a finite stream* $\mathcal{B} = B_1, \ldots, B_k$ *and the secret key* SK *algorithm* S *outputs a new stream* $\mathcal{B}' = B'_1, \ldots, B'_k$ *where* $B'_i = (B_i, A_i)$.
- V *is the* verification *algorithm. For every* $(PK, SK) = G(1^n)$ *and* $\mathcal{B}' = S(SK, \mathcal{B})$, *it holds that* $V(PK, B'_1, \ldots, B'_i) = 1$ *for* $1 \leq i \leq k$.

Notice that we modeled the off-line property by the fact that the signing algorithm is given the whole stream in advance. Yet the verifier is required to authenticate *each* prefix of the scheme without needing to see the rest of the stream. As it will become clear in the following our algorithms will not require the off-line verifier to store the whole past stream either.

Definition 2 *An* on-line stream signature scheme *is a triplet* (G, S, V) *of probabilistic polynomial-time algorithms satisfying the following properties:*

- G *is the* key generation *algorithm. On input* 1^n *it outputs a pair* $(SK, PK) \in \{0,1\}^{2n}$. SK *is called the secret (signing) key and* PK *is called the public (verification) key.*
- S *is the* signing *algorithm. Given a (possibly infinite) stream* $\mathcal{B} = B_1, \ldots,$ *algorithm* S *with input the secret key* SK *process each block one at the time, i.e.,*

$$S(SK, B_1, \ldots, B_i) = B'_i = (B_i, A_i)$$

- V *is the* verification *algorithm. For every* $(PK, SK) = G(1^n)$ *and* $B'_1, B'_2, \ldots$ *such that* $B'_i = S(SK, B_1, \ldots, B_i)$ *for all* i, *it holds that* $V(PK, B'_1, \ldots, B'_i) = 1$ *for all* i.

[2]The assumption that the blocks have all the same size is not really necessary. We just make it for clarity of presentation.

Notice that in the on-line definition we have the signer process each block "on the fly" so knowledge of future blocks is not needed. In this case also the definition seems to requires knowledge of all past blocks for both signer and verifier, however this does not have to be the case (indeed, in our solution some past blocks may be discarded).

The above definitions say nothing about security. In order to define security for stream signing we use the same notions of security introduced in [13]. That is, we require that no efficient algorithm will be able to produce a valid signed stream, even after seeing several signed streams. However, notice that given our definition of signed streams, a prefix of a valid signed stream is itself a valid signed stream. So the forger can present a "different" signed stream by just taking a prefix of the ones seen before. However, this hardly constitutes forgery, so we rule it out in the definition. With $\mathcal{B}^{(1)} \subseteq \mathcal{B}^{(2)}$ we denote the fact that $\mathcal{B}^{(1)}$ is a prefix of $\mathcal{B}^{(2)}$.

Definition 3 *We say that an off-line (resp. on-line) stream signature scheme (G, S, V) is* secure *if any probabilistic polynomial time algorithm F, given as input the public key PK and adaptively chosen signed streams $\mathcal{B}'^{(j)}$ for $j = 1, 2, \ldots$, outputs a new previously unseen valid signed stream $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}$ $\forall j$ only with negligible probability.*

For signed streams we slightly abuse the notation: when we write $\mathcal{B}'^{(1)} \not\subseteq \mathcal{B}'^{(2)}$ we mean that not only $\mathcal{B}'^{(1)}$ is not a prefix of $\mathcal{B}'^{(2)}$ but also the underlying "basic" unsigned streams are in the same relationship, i.e., $\mathcal{B}^{(1)} \not\subseteq \mathcal{B}^{(2)}$.

This is the definition of existential unforgeability against adaptively-chosen message attack, the strongest of the notions presented in [13]. Following [13] weaker variants can be defined. Notice that the adversary is polynomially bounded, thus he can query only a bounded number of streams whose total length will also be bounded by a polynomial.

4 The Off-Line Solution

In this case we assume that the sender knows the entire stream in advance. (e.g., music/movie broadcast). Assume for simplicity that the stream is such that it is possible to reserve 20 bytes of extra authentication information in a block of size c.

The stream is logically divided into blocks of size c. The receiver has a buffer of size c. The receiver first receives the signature on the 20 byte hash (e.g., SHA-1) of the first block. After verification of the signature the receiver knows what the hash of the first block should be and then starts receiving the full stream and starts computing its hash block by block. When the receiver receives the first block, it checks its hash against what the signature was verified upon. If it matches, it plays the block otherwise it rejects it and stops playing the stream. How are other blocks authenticated? The key point is that the first block contains the 20 byte hash of the second block, the second block contains the 20 byte hash of the third block and so on... Thus, after the first signature check, there are just hashes to be checked for every subsequent block.

In more detail, let (G, S, V) be a regular signature scheme. The sender has a pair of secret-public key $(SK, PK) = G(1^n)$ of such signature scheme. Also let H be a collision-resistant cryptographic hash function. If the original stream is

$$\mathcal{B} = B_1, B_2, \ldots, B_k$$

and the resulting signed stream is

$$B' = B'_0, B'_1, B'_2, \ldots, B'_k$$

the processing is done *backwards* on the original stream as follows:

$$B'_k =< B_k, 00\ldots0 >$$

$$B'_i =< B_i, H(B'_{i+1}) > \text{ for } i = 1, \ldots, k-1$$

$$B'_0 =< H(B'_1, k), S(SK, H(B'_1, k)) >$$

Notice that on the sender side, computing the signature and embedding the hashes requires a single *backwards* pass on the stream, hence the restriction that the stream is fully known in advance. Notice also that the first block B'_0 of the signed stream contains an encoding of the length of the stream (k).

The receiver verifies the signed stream as follows: on receiving $B'_0 =< B, A_0 >$ she checks that

$$V(PK, A_0, B) = 1$$

and extracts the length k in blocks of the stream (which we may assume is encoded in the first block). Then on receiving $B'_i =< B_i, A_i >$ (for $1 \leq i \leq k$) the receiver accepts B_i if

$$H(B'_i) = A_{i-1}$$

Thus the receiver has to compute a single public-key operation at the beginning, and then only one hash evaluation per block. Notice that no big table is needed in memory.

5 The On-Line Solution

In this case the sender does not know the entire stream in advance (e.g, live broadcast). In this scenario it is important that also the operation of signing (and not just verification) be fast, since the sender himself is bound to produce an authenticated stream at a potentially high rate.

ONE-TIME SIGNATURES. In the following we will use a special kind of signature scheme introduced in [15, 16]. These are signatures which are much faster to compute and verify than regular signatures since they are based on one-way functions and do not require a trapdoor function. Conjectured known one-way functions (as DES or SHA-1) are much more efficient than the known conjectured trapdoor functions as RSA. However, these schemes cannot be used to sign an arbitrary number of messages but only a prefixed number of them (usually one). Several other 1-time schemes have been proposed [12, 6, 7]; in Section 7 we discuss possible instantiations for our purpose.

In this case also the stream is split into blocks. Initially the sender sends a signed public key for a 1-time signature scheme. Then he sends the first block along with a 1-time signature on its hash based on the 1-time public key sent in the previous block. The first block also contains a new 1-time public key to be used to verify the signature on the second block and this structure is repeated in all the blocks.

More in detail: let us denote with (G, S, V) a regular signature scheme and with (g, s, v) a 1-time signature scheme. With H we still denote a collision-resistant hash function. The sender has long-lived keys $(SK, PK) = G(1^n)$. Let

$$\mathcal{B} = B_1, B_2, \ldots$$

be the original stream (notice that in this case we are not assuming the stream to be finite) and

$$\mathcal{B}' = B'_0, B'_1, B'_2, \ldots$$

the signed stream constructed as follows. For each $i \geq 1$ let us denote with $(sk_i, pk_i) = g(1^n)$ the output of an independent run of algorithm g. Then

$$B'_0 =< pk_0, S(SK, pk_0) >$$

(public keys of 1-time signature schemes are usually short so they need not to be hashed before signing)

$$B'_i =< B_i, pk_i, s(sk_{i-1}, H(B_i, pk_i)) > \text{ for } i \geq 1$$

Notice that apart from a regular signature on the first block, all the following signatures are 1-time ones, thus much faster to compute (including the key generation, which does not have to be done on the fly.)

The receiver verifies the signed stream as follows. On receiving $B'_0 =< pk_0, A_0 >$ she checks that

$$V(PK, A_0, pk_0) = 1$$

and then on receiving $B'_i =< B_i, pk_{i+1}, A_i >$ she checks that

$$v(pk_{i-1}, A_i, H(B_i, pk_i)) = 1$$

whenever one of these checks fails, the receiver stops playing the stream. Thus the receiver has to compute a single public-key operation at the beginning, and then only one 1-time signature verification per block.

6 Proofs of Security

We are able to prove the security of our stream signature schemes according to the definitions presented in Section 3, provided that the underlying components used to build the schemes are secure on their own.

THE OFF-LINE CASE. Let us denote with $(\mathcal{G}_{off}, \mathcal{S}_{off}, \mathcal{V}_{off})$ the off-line stream signature scheme described in Section 4. With (G, S, V) let us denote the "regular" signature scheme and with H the hash function used in the construction. The following holds.

Theorem 1 *If (G, S, V) is a secure signature scheme and H is a collision-resistant hash function then the resulting stream signature scheme $(\mathcal{G}_{off}, \mathcal{S}_{off}, \mathcal{V}_{off})$ is secure.*

Proof

Assume the thesis is false, i.e., that there is an algorithm $\mathcal{F}$ that succeeds in an existential forgery against $(\mathcal{G}_{off}, \mathcal{S}_{off}, \mathcal{V}_{off})$ using an adaptively-chosen message attack with non-negligible probability ϵ. That is, $\mathcal{F}$ runs on input PK, adaptively asks for the signed versions of ℓ streams $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(\ell)}$ where $\mathcal{B}^{(j)} = B_1^{(j)} \ldots B_{k_j}^{(j)}$ and receives them; let them be $\mathcal{B}'^{(1)}, \ldots, \mathcal{B}'^{(\ell)}$. Then $\mathcal{F}$ outputs a valid signed stream $\mathcal{B}'$ where $\mathcal{B}' = B'_0 \ldots B'_k$ is not a prefix of any of the previous ones, i.e., $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}\ \forall j = 1, \ldots, \ell$. One of the following two cases must be true.

Case 1: *With probability at least $\epsilon/2$, $\mathcal{F}$ outputs a signed stream whose first block coincides with the first block of one of the signed streams it asked for before.* That is there exists a j such that $B'_0 = B'^{(j)}_0$. We show in this case how $\mathcal{F}$ can be used to build an algorithm $\mathcal{F}_c$ which finds collisions for H.

$\mathcal{F}_c$ runs as follows. It first runs G to obtain a pair of public and secret key (PK, SK). Then it runs $\mathcal{F}$ and uses SK to sign all the requests $\mathcal{B}^{(j)}$ for $j = 1, \ldots, \ell$ that $\mathcal{F}$ makes. Now with probability at least $\epsilon/2$ $\mathcal{F}$ outputs $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}$ for all j, but such that there exists j such that $B'_0 = B'^{(j)}_0$. Let k be the length of the stream $\mathcal{B}'$ and k_j the length of the stream $\mathcal{B}'^{(j)}$.

If $k \neq k_j$ then $(B'_1, k) \neq (B'^{(j)}_1, k_j)$. But since $B'_0 = B'^{(j)}_0$ it must be that $H(B'_1, k) = H(B'^{(j)}_1, k_j)$. So $\mathcal{F}_c$ can output the pair $(B'_1, k), (B'^{(j)}_1, k_j)$ as a collision for H.

If $k = k_j$, then since $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}$ there must exist an $0 < \alpha \leq k$ such that $B'_\alpha \neq B'^{(j)}_\alpha$ while $B'_\beta = B'^{(j)}_\beta$ for all $0 \leq \beta < \alpha$. That is we have that $B'_{\alpha-1} = B'^{(j)}_{\alpha-1}$, which (by our construction of signed streams) implies that $H(B'_\alpha) = H(B'^{(j)}_\alpha)$. So $\mathcal{F}_c$ outputs the pair $B'_\alpha, B'^{(j)}_\alpha$ as a collision for H.

Case 2: *With probability at least $\epsilon/2$, $\mathcal{F}$ outputs a signed stream whose first block is different from the first block of any of the signed streams it asked for before.* That is, for all j we have $B'_0 \neq B'^{(j)}_0$. We show in this case how $\mathcal{F}$ can be used to build an algorithm $\mathcal{F}_s$ which forges signatures for the regular signature scheme (G, S, V).

$\mathcal{F}_s$ runs on input a public key PK of the signature scheme (G, S, V). It is also allowed to query a signature oracle to get signed messages of its choice. $\mathcal{F}_s$ starts by running $\mathcal{F}$. When the latter requests a signature on a stream $\mathcal{B}^{(j)} = B_1^{(j)} \ldots B_{k_j}^{(j)}$ $\mathcal{F}_s$ first prepares the last k_j blocks of the signed stream $B'^{(j)}_1, \ldots, B'^{(j)}_{k_j}$ and then it uses its allowed queries to the signature oracle to compute $B'^{(j)}_0 =< H(B'^{(j)}_1), S(SK, H(B'^{(j)}_1)) >$.

Now with probability at least $\epsilon/2$ $\mathcal{F}$ outputs $\mathcal{B}' = B'_0 \ldots B'_k$ such that $B'_0 \neq B'^{(j)}_0$ for all j. This in turn means that B'_0 is a message/signature pair that $\mathcal{F}_s$ has not asked before to the signature oracle. $\mathcal{F}_s$ stops outputting such pair as the forged signature.

Since $\epsilon/2$ is a non-negligible probability, either Case 1 or Case 2 contradicts our hypothesis, so the thesis must be true.

■

THE ON-LINE CASE. Let us denote with $(\mathcal{G}_{on}, \mathcal{S}_{on}, \mathcal{V}_{on})$ the on-line stream signature scheme described in Section 5. With (G, S, V) let us denote the "regular" signature scheme, with

(g, s, v) the one-time signature scheme and with H the hash function used in the construction. The following holds.

Theorem 2 *If (G, S, V) and (g, s, v) are secure signature schemes and H is a collision-resistant hash function then the resulting stream signature scheme $(\mathcal{G}_{on}, \mathcal{S}_{on}, \mathcal{V}_{on})$ is secure.*

Proof

Assume the thesis is false, i.e., that there is an algorithm $\mathcal{F}$ that succeeds in an existential forgery against $(\mathcal{G}_{on}, \mathcal{S}_{on}, \mathcal{V}_{on})$ using an adaptively-chosen message attack with non-negligible probability ϵ. That is $\mathcal{F}$ runs on input PK, adaptively asks for the signed versions of ℓ streams[3] $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(\ell)}$ where $\mathcal{B}^{(j)} = B_1^{(j)} \ldots$ and receives them; let them be $\mathcal{B}'^{(1)}, \ldots, \mathcal{B}'^{(\ell)}$. Then $\mathcal{F}$ outputs a valid signed stream $\mathcal{B}'$ where $\mathcal{B}' = B'_0, \ldots$ which is not a prefix of any of the previous ones, i.e., $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}$ $\forall j = 1, \ldots, \ell$. One of the following cases must be true.

Case 1: *With probability at least $\epsilon/2$, $\mathcal{F}$ outputs a signed stream whose first block is different from the first block of any of the signed streams it asked for before.* That is, for all j we have $B'_0 \neq B'^{(j)}_0$. We show in this case how $\mathcal{F}$ can be used to build an algorithm $\mathcal{F}_1$ which forges signatures for the regular signature scheme (G, S, V).

$\mathcal{F}_1$ runs on input a public key PK of the signature scheme (G, S, V). It is also allowed to query a signature oracle to get signed messages of its choice. $\mathcal{F}_1$ starts by running $\mathcal{F}$. When the latter requests a signature on a stream $\mathcal{B}^{(j)} = B_1^{(j)} \ldots$, $\mathcal{F}_s$ prepares the first block of the signed stream $B'^{(j)}_0$ by running g to get $(pk_0^{(j)}, sk_0^{(j)})$ and then queries the signature oracle to get a signature on pk_0. The remaining blocks of the stream can be easily prepared by $\mathcal{F}_1$ by running the 1-time key generation algorithm g as needed

Now with probability at least $\epsilon/2$ $\mathcal{F}$ outputs $\mathcal{B}' = B'_0 \ldots$ such that $B'_0 \neq B'^{(j)}_0$ for all j. This in turn means that the B'_0 is a message/signature pair for (G, S, V) that $\mathcal{F}_1$ has not asked before to the signature oracle. $\mathcal{F}_1$ stops outputting such pair as the forged signature.

Case 2: *With probability at least $\epsilon/2$, $\mathcal{F}$ outputs a signed stream whose first block coincides with the first block of one of the signed streams it asked for before.* That is there exists a j such that $B'_0 = B'^{(j)}_0$. However, we also know that $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}$. This implies that there exists an $0 < \alpha < k$ such that $B'_\alpha \neq B'^{(j)}_\alpha$ while $B'_{\alpha-1} = B'^{(j)}_{\alpha-1}$. Recall that, by our construction of signed streams, we have

$$B'_\alpha = < B_\alpha, pk_\alpha, s(sk_{\alpha-1}, H(B_\alpha, pk_\alpha)) >$$

$$B'^{(j)}_\alpha = < B^{(j)}_\alpha, pk^{(j)}_\alpha, s(sk^{(j)}_{\alpha-1}, H(B^{(j)}_\alpha, pk^{(j)}_\alpha)) >$$

We know that $B_\alpha \neq B^{(j)}_\alpha$. One of this two subcases must be true.

Case 2a: *With probability at least $\epsilon/4$ the output of $\mathcal{F}$ is such that*

$$H(B_\alpha, pk_\alpha) = H(B^{(j)}_\alpha, pk^{(j)}_\alpha).$$

We show how to construct an algorithm $\mathcal{F}_2$ that computes collisions for H.

[3] We may assume that the forger adaptively chooses the components of the stream, i.e., after seeing the i^{th} block signed it creates the $(i+1)^{st}$ block to be signed

$\mathcal{F}_2$ runs as follows. It first runs $\mathcal{G}$ to obtain a pair of public and secret key (PK, SK). Then it runs $\mathcal{F}$. When the latter asks to sign a specific stream, $\mathcal{F}_2$ uses SK to sign the first block and then generates "on the fly" one-time keys (using g) to sign all the other blocks of the given stream. Then $\mathcal{F}_2$ looks at the output of $\mathcal{F}$ which by assumption has the property that

$$H(B_\alpha, pk_\alpha) = H(B_\alpha^{(j)}, pk_\alpha^{(j)})$$

and since we know that $B_\alpha \neq B_\alpha^{(j)}$, $\mathcal{F}_2$ has found a collision.

Case 2b: *With probability at least $\epsilon/4$ the output of $\mathcal{F}$ is such that*

$$H(B_\alpha, pk_\alpha) \neq H(B_\alpha^{(j)}, pk_\alpha^{(j)}).$$

We show how to construct an algorithm $\mathcal{F}_3$ that forges signatures in the 1-time scheme (g, s, v).

Let us denote with K an upper bound on the total number of stream blocks that $\mathcal{F}$ asks during its attempt at forgery. Without loss of generality let's assume that $\mathcal{F}$ always asks K blocks. Clearly T is polynomial in our security parameter n.

$\mathcal{F}_3$ works as follows. It runs on input a 1-time key pk and it can ask one query to get a single message signed by the corresponding secret key sk. Its goal is to output a different message and its signature under sk.

$\mathcal{F}_3$ runs as follows. It runs G to obtain a long-lived key pair PK, SK of (G, S, V). It then runs g several times in order to obtain several one-time key pairs. Finally it selects uniformly at random an integer i between 1 and $K-1$.

$\mathcal{F}_3$ starts running $\mathcal{F}$. Whenever the latter asks for a signed stream, $\mathcal{F}_3$ can sign the first block since it knows SK and it uses the generated 1-time keys for the internal blocks. The exception is when $\mathcal{F}$ asks for the i^{th} block (sequentially). In this case $\mathcal{F}_3$ uses pk as the 1-time public key embedded in the i^{th} block. This means that when $\mathcal{F}$ asks for the $(i+1)^{st}$ block, $\mathcal{F}_3$ has to query the signature oracle in order to compute the 1-time signature embedded in it.

At the end of this process, $\mathcal{F}$ stops outputting a signed stream with the properties outlined above. With probability $1/K$ the block $B'^{(j)}_{\alpha-1}$ is the one in which $\mathcal{F}_3$ used the target public key pk. This in turn means that the 1-time signature contained in the block B'_α outputted by $\mathcal{F}$ is valid under the key pk, yet it is on a different message than the one queried by $\mathcal{F}_3$.

So with probability $\epsilon/4K$, which is still non-negligible, the forger $\mathcal{F}_3$ succeeds.

■

Remark: The statements of the above theorems are valid not only in asymptotic terms, but have also a concrete interpretation which ultimately is reflected in the key lengths used in the various components in order to achieve the desired level of security of the full construction. It is not hard to see, by a close analysis of the proofs, that the results are pretty tight. That is, a forger for the stream signing scheme can be transformed into an attacker for one of the components (the hash function, the regular signature scheme and, a little less optimally, the 1-time signature scheme) which runs in about the same time, asks the same number of queries and has almost the same success probability. This is turns

means that there is no major degradation in the level of security of the compound scheme and thus the basic components can be run with keys of ordinary length.

More precisely: for the off-line scheme, if we have a forger $\mathcal{F}$ that runs in time T, asks for q signed streams of total length K and succeeds with probability ϵ then we have either a collision-finder $\mathcal{F}_c$ for $\mathcal{H}$ or a forger $\mathcal{F}_s$ for the regular signature scheme which run in time T, ask q signature queries and qK hashing queries and succeed with probability $\epsilon/2$.

For the on-line scheme, if we have a forger $\mathcal{F}$ that runs in time T, asks for q signed streams of total length K and succeeds with probability ϵ then we have

- either a forger $\mathcal{F}_1$ for the regular signature scheme which runs in time $T + Kt$, asks q signature queries and qK hashing queries and succeed with probability $\epsilon/2$.
- or a collision-finder $\mathcal{F}_2$ for $\mathcal{H}$ which runs in time $T + Kt$, asks q signature queries and qK hashing queries and succeed with probability $\epsilon/4$.
- or a forger $\mathcal{F}_3$ for the one-time signature scheme which runs in time $T + Kt$, asks q signature queries and qK hashing queries and succeed with probability $\epsilon/4K$.

where t is the time required to run the one-time signature scheme (key generation and signature) algorithms.

7 Implementation Issues

7.1 The Choice of the One-Time Signature Scheme

Several one-time schemes have been presented in the literature, see for example [15, 16, 12, 6, 7]. The main parameters of these schemes are signature length and verification time. In the solutions we know of, these parameters impose conflicting requirements, i.e., if one wants a scheme with short signatures, verification time goes up, while schemes with longer signatures can have a much shorter verification time. In our on-line solution we would like to keep both parameters down. Indeed, the verification should be fast enough to allow the receiver to consume the stream blocks at the same input rate she receives them. At the same time, since the signatures are embedded in the stream, it's important to keep them small so that they will not reduce the throughput rate of the original stream.

We first suggest a scheme which obtain a reasonable compromise. The scheme is based on a 1-way function F in a domain D. It also uses a collision resistant hash function H. The scheme allows signing of a single m-bit message. It is based on a combinations of ideas from [15, 25]. Here are the details of the scheme.

Key Generation. Choose $m + \log m$ elements in D, let them be $a_1, \ldots, a_{m+\log m}$. This is the secret key. The public key is

$$pk = H(F(a_1), \ldots, F(a_{m+\log m}))$$

Signing Algorithm. Let M be the message to be signed. Append to M the binary representation of the number of zeros in M's binary representation. Call M' the resulting binary string. The signature of M is $s_1, \ldots, s_{m+\log m}$ where $s_i = a_i$ if the i^{th} bit of M' is 1 otherwise $s_i = F(a_i)$.

Verification Algorithm. Check if

$$H(t_1, \ldots, t_{m+\log m}) = pk$$

where $t_i = s_i$ if i^{th} bit of M' is 0 otherwise $t_i = F(s_i)$.

Security. Intuitively this scheme is secure since it is not possible to change a 0 into a 1 in the binary representation of the message M without having to invert the function F. It is possible to change a 1 into a 0, but that will increase the number of 0's in the binary representation of M causing a bit to flip from 0 to 1 in the last $\log m$ bits of M', and so forcing the attacker to invert F anyway.

Parameters. This scheme has signature length $|D|(m + \log m)$ where $|D|$ is the number of bits required to represent elements of D. The receiver has to compute 1 hash computation of H plus on the average $\frac{m+\log m}{2}$ computations of F.

In practice we assume that F maps 64-bit long strings into 64-bit long strings. Since collision resistance is not required from F we believe this parameter to be sufficient. Conjectured good F's can be easily constructed from efficient block ciphers like DES or from fast hash functions like MD5 or SHA-1. [4] Similarly H can be instantiated to MD5 or SHA-1. In general we may assume m to be 128 or 160 if the message to be signed is first hashed using MD5 or SHA-1.

The SHA-1 implementation has then signatures which are 1344 bytes long. The receiver has to compute F around 84 times on the average. With MD5 the numbers become 1080 bytes and 68 respectively. When used in our off-line scheme one also has to add 16 bytes for the embedding of the public key in the stream.

Remark: Comparing the RSA signature scheme with verification exponent 3 with the above schemes, one could wonder if the verification algorithm is really more efficient (2 multiplications versus 84 hash computations). Typical estimates today are that an RSA verification is comparable to 100 hash computations. However, we remind the reader that we are trying to improve *both* signature generation and verification. Indeed, this scheme is used in the on-line case and as such both operations have to be performed on-line and thus efficiently. The improvement in signature generation is much more substantial.

Other schemes: The scheme above could be improved in the length of the signature by using Winternitz's idea [25]. The public key is composed of $m + 1$ elements of D, let them be $a_0, a_1, \ldots, a_m$. The public key is $pk = H(F^m(a_0), F(a_1), \ldots, F(a_m))$. The signature on a message M is $s_0, s_1, \ldots, s_m$ where for $i \geq 1$, $s_i = a_i$ if the i^{th} bit of M' is 1 otherwise $s_i = F(a_i)$, while $s_0 = F^{\ell}(a_0)$ where ℓ is the number of 1's in M's binary representation. The verification of the signature takes the same amount of computation as the scheme described above. However, the length of the signature is slightly shorter, i.e., $|D|(m + 1)$ bits. For a SHA-1 implementation this means 1288 bytes. As noticed in [12] the security of this scheme is based on a somewhat stronger assumption on F. Namely F is assumed to be *non-quasi-invertible*, that is on input y it infeasible to find an index i and a value

[4] As a cautionary remark to prevent attacks where the attacker builds a large table of evaluations of F, in practice F could be made different for each signed stream (or for each large portion of the signed stream) by defining $F(x)$ to be $G(Salt||X)$ where G is a one-way 128 bit to 64 bit function, and the *Salt* is generated at random by the signer once for each stream or large pieces thereof.

x such that $F^{i+1}(x) = F^i(y)$. Clearly if F is a one-way permutation the above notion is automatically satisfied.

This scheme can be further generalized as described in [12]. The message M is split in $\frac{m}{t}$ blocks of size t bits, let them be $M_1, \ldots, M_{\frac{m}{t}}$. The secret keys are composed of $\frac{m}{t} + 1$ elements of D, let them be $a_0, a_1, \ldots, a_{\frac{m}{t}}$. The public key is

$$pk = H(F^{(2^t-1)\frac{m}{t}}(a_0), F^{2^t-1}(a_1), \ldots, F^{2^t-1}(a_{\frac{m}{t}})).$$

The signature of a message M is computed by considering the integer value of the blocks M_i. The signature is composed by $\frac{m}{t}+1$ values $s_0, s_1, \ldots, s_{\frac{m}{t}}$ where, for $i \geq 1$, $s_i = F^{2^t-1-M_i}(x_i)$, while $s_0 = F^{\sum_i M_i}(x_0)$. In this case the signature length is $|D|(\frac{m}{t} + 1)$ bits, but verification time goes up with the parameter t. Indeed, the number of hash computations goes as $O(\frac{2^t m}{t})$. Also this scheme is based on the non-quasi-invertibility of F.

In general one has to look at the specific application and decide among the tradeoffs specified by the parameter t in order to decide if it is better to reduce the signature length or the computation time. See also Section 7.3 for other ways to deal with this issue.

7.2 Non-Repudiation

In case of a legal dispute over the content of a signed stream the receiver must bring to court some evidence. If the receiver saves the whole stream, then there is no problem. However, in some cases, for example because of memory limitations, the receiver might be forced to discard the stream data after having consumed it. In these cases what should she save to protect herself in case of a legal dispute?

In the off-line solution, assuming the last block of the signed stream always has a special reserved value for the hash-chaining field, (say all 0's) she needs to save only the first signed block. Indeed, this proves that she received something from the sender. Now we could conceivably move the burden of proof to the sender to reconstruct the whole stream that matches that first block and ends with the last block which has the reserved value for the hash-chaining field (a similar idea was used in [1] to deal with storage limitations on low-end devices).

Similarly in the on-line solution, at a minimum the receiver needs to save the first signed block and all 1-time signatures and have the sender reconstruct the stream. However, in practice, this may still be too much to save. For practical applications, we suggest the following scheme. The on-line stream could be broken up into a sequence of chunks, each chunk representing a logical unit, e.g., a TV programme or a live broadcast of a game or even programming sent over some fixed sized time interval. The idea is that once a logical unit is decided, an upper bound on the total number of blocks in it can be estimated and all the one-time keys needed for a chunk of that size could be precomputed by the sender. In addition, by using the off-line stream signing technique, the sender can compute a single hash value which when signed can be used to authenticate each of these one-time public keys if they were to be sent as a stream of keys, one key in each stream block. The new on-line scheme works as follows: Initially the sender sends a regular digital signature on the hash value which can be used to authenticate a stream of one-time public keys. The stream of one-time keys is then embedded in the actual on-line data stream and the data

stream itself is authenticated by one-time signatures based on the one-time keys where each one-time signature is on the running hash of all the data sent so far on the stream. This way the receiver only needs to store the initial regular digital signature and the last one-time signature that was received and verified. For non-repudiation purposes, based on the initial regular digital signature, the sender can be forced to disclose the entire stream of one-time public keys, and, based on the last valid one-time signature stored by the receiver, the sender can be forced to produce a data stream with the same hash as what is signed by the last one-time signature stored by the receiver.

7.3 Hybrid Schemes

In the on-line scheme, the length of the embedded authentication information is of concern as it could cut into the throughput of the stream. In order to reduce it, hybrid schemes can be considered. In this case we assume that some asynchrony between the sender and receiver is acceptable.

Suppose the sender can process a group (say 20) of stream blocks at a time before sending them. With a pipelined process this would only add an initial delay before the stream gets transmitted. The sender will sign with a one-time key only 1 block out of 20. The 20 blocks in between these two signed blocks will be authenticated using the off-line scheme. This way the long 1-time signatures and the verification time can be amortized over the 20 blocks.

A useful feature of our proposed 1-time signature scheme is that it allows the verification of (the hash of) a message bit by bit. This allows us to actually "spread out" the signature bits and the verification time among the 20 blocks. Indeed, if we assume that the receiver is allowed to play at most 20 blocks of unauthenticated information before stopping if tampering is detected we can do the following. We can distribute the signature bits among the 20 blocks and verify the hash of the first block bit by bit as the signature bits arrive. This maintains the stream rate stable since we do not have long signatures sent in a single block and verification now takes 3-4 hash computations per block, on *every* block.

It is also possible to remove the constraint on playing 20 blocks of unauthenticated information before tampering is detected. This requires a simple modification to our on-line scheme. Instead of embedding in block B_i its own 1-time signature, we embed the signature of the next block B_{i+1}. This means that in the on-line case blocks have to be processed two at a time now. When this modification is applied to the hybrid scheme, the signature bits in the current 20 blocks are used to authenticate the following 20 blocks so unauthenticated information is never played. However, this means that now the sender has to process 40 blocks at a time in the hybrid scheme.

7.4 Probabilistic One-Time Signatures.

The length and the computation time of a 1-time signature are determined by the length of the message being signed. Typically the message is first hashed, thus the range of the collision-resistant hash function is the crucial parameter here. However, in order to make sure that the function is a strong collision-resistant one, it is necessary to assume a long range. MD5 with 128-bits is considered borderline. SHA-1 seems to give more security with a 160-bits range.

Is it possible to use a weak collision resistant hash function to hash messages instead? In the presence of a chosen message attack this is not possible as an attacker could find two message x and y that hash to the same value and by obtaining a signature on x would then obtain a signature on y. It is however possible to foil this particular attacks by first randomizing the message being signed.

Consider the following approach. Let (G, S, V) be a secure signature scheme (against adaptively-chosen message attack) on a message of size b, but we want to sign messages of arbitrary length. Let H_k be a family of weak collision resistant hash functions whose range is $(b - |k|)$-bit strings. Consider the following new signature scheme on messages of arbitrary size. On input a message M, choose a random k, compute $h = H_k(M)$ and output k together with a signature on the pair k, h. We claim that this new signature scheme is also secure against adaptively-chosen message attack. This is because the receiver cannot use the fact that H_k is a weak collision hash function, because she does not know which function is going to be used.

The only issue here is non-repudiation as the signer can find collisions and when it is challenged with a signed message M he can present another message M' that has the same signature. However, this is not really a problem as one can hold the signer responsible for *any* signed message as only he can find collisions. A more detailed discussion about these issues can be found in [3].

Notice that the family H_k can be easily built out of regular hash functions via "keying" techniques. For example in iterative constructions of H, k could be used as the IV. In practice we suggest to use the first 80 bits of SHA-1 keyed via the IV with k.

The above technique does not have a particular impact with typical signature schemes as reducing the range of the hash function is not an issue there. But with 1-time signatures this allows for some savings in the length of the signature and the computation time. For example with typical lengths $|k|$ would be say 60 bits and a weak collision hash function would be around 80 bits in range (to obtain a level of security comparable to the 160-bit strong collision-resistant functions). So the probabilistic hashing would return a value of 140 bits which implies a savings of almost 15% in length and computation time. We stress that this a general result for all 1-time signature schemes.

When applied to our stream signature scheme that improvement itself is already valuable. But in our specific case we can improve even more as we can use the hybrid approach once again. Indeed, we don't have to use a different k for each block. The signer could keep k fixed for a limited amount of time. This time window should be small enough to prevent an attacker from finding collisions for the hash function H_k which is kept fixed during this time window. If H_k is a good hash function it should take roughly 2^{40} steps to find such collisions. Thus a small window of a few seconds should not pose security problems. If the rate of the stream is high enough during this time window we will sign several blocks. Say for simplicity that we sign 20 blocks with the same k. This will allow us to spread the bits of k on 20 blocks which means that now the probabilistic hashing returns an 83-bit value per block, thus achieving almost a 50% efficiency improvement in signatures length and computation time!

Remark: In order to use this solution and be able to preserve non-repudiation the recipient *must* save the whole stream, i.e., the techniques described in Section 7.2 cannot be applied.

Indeed, those techniques rely on the fact that the signer cannot find a collision and thus repudiate the message he sent to the receiver (in case the latter did not save the original message, but just its hash).

8 Applications

MPEG VIDEO AND AUDIO. In the case of MPEG video and audio, there are several methods for embedding authentication data. Firstly, the Video Elementary stream has a USER-DATA section where arbitrary user defined information can be placed. Secondly, the MPEG system layer allows for an elementary data stream to be multiplexed synchronously with the packetized audio and video streams. One such elementary stream could carry the authentication information. Thirdly, techniques borrowed from digital watermarking can be used to embed information in the audio and video itself at the cost of slight quality degradation. In the case of MPEG video since each frame is fairly large, (hundreds of kilobits) and the receiver is required to have a buffer of at least 1.8Mbits, both the off-line as well as the on-line solutions can be deployed without compromising picture quality. In the case of audio however, in the extreme case the bit rate could be very low (e.g., 32Kbits/s) and each audio frame could be small (approx. 1000 bytes) and the receiver's audio buffer may be tiny (less than 2 Kbytes). In such extreme cases the on-line method, which requires around 1000 bytes of authentication information per block cannot be used without seriously cutting into audio quality. For these extreme cases, the best on-line strategy would be to send the authentication information via a separate but multiplexed MPEG data stream. For regular MPEG audio, if the receiver has a reasonably sized buffer (say 32K) then we can apply the on-line scheme with a large block size (say 20 K) to obtain a signed MPEG audio stream with a delay of approximately 5-6 seconds and space overhead of 5%. If the receiver buffer is small but not tiny (say 3 K) a hybrid scheme would work: for example one could use groups of 33 hash–chained blocks of length 1000 bytes each; this would typically result in a 5% degradation and a delay in the 20 second range.

JAVA. In the original version of java (JDK 1.0), for an applet coming from the network, first the startup class was loaded and then additional classes were (down) loaded by the class loader in a lazy fashion as and when the running applet first attempted to access them. Since our ideas apply not only to streams which are a linear sequence of blocks but in general to trees as well (where one block can invoke any of its children), based on our model, one way to sign java applets would be to sign the startup class and each downloaded class would have embedded in it the hashes of the additional classes that it downloads directly. However, for code signing, Javasoft has adopted the multiple signature and hash table based approach in JDK1.1, where each applet is composed of one or several Java archives, each of which contains a signed table of hashes (the manifest) of its components. It is our belief that once java applets become really large and complex the shortcomings of this approach will become apparent: (1) the large size of the hash table in relation to the classes actually invoked during a run. This table has to be fully extracted and authenticated before any class gets authenticated; (2) the computational cost of signing each of the manifests if an applet is composed of several archives; (3) accommodating classes or data resources which are generated on the fly by the application server based on a client request.

These could be addressed by using some of our techniques. Also the problem of how to sign audio/video streams will have to be considered in the future evolution of Java, since putting the hash of a large audio/video file in the manifest would not be acceptable.

BROADCAST APPLICATIONS. Our schemes (both the off-line and the on-line one) can be easily modified to fit in a broadcast scenario. Assume that the stream is being sent to a broadcast channel with multiple receivers who dynamically join or leave the channel. In this case a receiver who joins when the transmission is already started will not be able to authenticate the stream since she missed the first block that contained the signature. Both schemes however can be modified so that every once in a while apart from the regular chaining information, there will also be a regular digital signature on a block embedded in the stream. Receivers who are already verifying the stream via the chaining mechanism can ignore this signature whereas receivers tuned in at various time will rely on the first such signature they encounter to start their authentication chain.

A different method to authenticate broadcasted streams, with weaker non-repudiation properties than ours, is proposed in [5].

LONG FILES WHEN COMMUNICATION IS COSTLY. Our solution can be used also to authenticate long files in a way to reduce communication cost in case of tampering. Suppose that a receiver is downloading a long file from the Web. There is no "stream requirement" to consume the file as it is downloaded, so the receiver could easily receive the whole file and then check a signature at the end. However, if the file has been tampered with, the user will be able to detect this fact only at the end. Since communication is at a cost (time spent online, bandwidth wasted etc) this is not a satisfactory solution. Using our solution the receiver can interrupt the transmission as soon as tampering is detected thus saving precious communication resources.

SIGNATURE-BASED CONTENT FILTERING AT PROXIES. Recently there has been interest in using digital signatures as a possible way to filter content admitted in by proxy servers through firewalls. Essentially when there is a firewall and one wishes to connect to an external server, then this connection can only be done via a proxy server. In essence one establishes a connection to a proxy and the proxy establishes a separate connection to the external server (if that is permitted). The proxy then simulates a connection between the internal machine and the external machine by copying data between the two connections. There has been some interest in modifying proxies so that they would only allow signed data to flow from the external server to the internal machine. However, since the proxy is only copying data as it arrives from the external connection into the internal connection and it cannot store all the incoming data before transferring it, the proxy cannot use a regular signature scheme for solving this problem. However, it is easy to see that in the proxy's view the data is a stream. Hence if there could be some standardized way to embed authentication data in such streams, then techniques from this paper would prove useful in solving this problem.

9 Acknowledgments

We would like to thank Hugo Krawczyk for his advice, guidance and encouragement for this work. We would also like to thank Mihir Bellare, Ran Canetti and Mike Wiener for helpful

discussions.

References

[1] N. Asokan, G. Tsudik, M. Waidner. Server-supported signatures. *ESORICS 96*, LNCS 1146, Springer-Verlag 1996, pp. 131-143.

[2] M. Bellare, S. Micali. How to Sign Given any Trapdoor Permutation. *J. of the ACM*, 39(1):214–233, 1992.

[3] M. Bellare, P. Rogaway. Collision-Resistant Hashing: Towards Making UOWHF's Practical. *Advances in Cryptology–CRYPTO'97*. LNCS, vol.1294, pp.470-484, Springer–Verlag, 1997.

[4] J. Benaloh, M. de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. *Advances in Cryptology–EUROCRYPT'93*. LNCS, vol.765, pp.274–285, Springer–Verlag, 1994.

[5] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. *IEEE INFOCOM'99*.

[6] D. Bleichenbacher, U. Maurer. Optimal Tree-Based One-time Digital Signature Schemes. *STACS'96*, LNCS, Vol. 1046, pp.363–374, Springer-Verlag.

[7] D. Bleichenbacher, U. Maurer. On the efficiency of one-time digital signatures. *Advances in Cryptology–ASYACRYPT'96*, to appear.

[8] I. Damgard. Collision Free Hash Functions and Public Key Signature Schemes. *Advances in Cryptology–EUROCRYPT'87*. LNCS, Springer–Verlag, 1988.

[9] I. Damgard. A Design Principle for Hash Functions. *Advances in Cryptology–Crypto'89*. LNCS, vol.435, Springer–Verlag, 1990.

[10] W. Diffie, M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):74–84, 1976.

[11] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[12] S. Even, O. Goldreich, S. Micali. On–Line/Off–Line Digital Signatures. *J. of Cryptology*, 9(1):35–67, 1996.

[13] S. Goldwasser, S. Micali, R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen Message Attack. *SIAM J. Comp.* 17(2):281–308, 1988.

[14] S. Haber, W. Stornetta. How to Time-Stamp a Digital Document. *Journal of Cryptology* 3(2):99–111, 1991.

[15] L. Lamport. Constructing Digital Signatures from a One-Way Function. *Technical Report SRI Intl.* CSL 98, 1979.

[16] R. Merkle. A Digital Signature based on a Conventional Encryption Function. *Advances in Cryptology–Crypto'87*. LNCS, vol.293, pp. 369–378, Springer–Verlag, 1988.

[17] R. Merkle. A Certified Digital Signature. *Advances in Cryptology–Crypto'89*. LNCS, vol.435, pp. 218–238, Springer–Verlag, 1990.

[18] R. Merkle. One-way Hash Functions and DES. *Advances in Cryptology–Crypto'89*. LNCS, vol.435, Springer–Verlag, 1990.

[19] National Institute of Standard and Technology. Secure Hash Standard. NIST FIPS Pub 180-1, 1995.

[20] M. Naor, M. Yung. Universal One-Way Hash Functions and their Cryptographic Applications. *Proceedings of STOC 1989*, pp.33–43.

[21] R. Rivest. The MD5 Message Digest Algorithm. Internet Request for Comments. April 1992.

[22] R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Comm. of the ACM*, 21(2):120–126, 1978.

[23] P. Rohatgi. A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication. *Proceedings of ACM CCS '99*, pp. 93–100.

[24] J. Rompel. One-Way Functions are Necessary and Sufficient for Secure Signatures. *Proceedings of STOC 1990*, pp.387–394.

[25] Winternitz. Personal communication to R. Merkle.

[26] C. Wong, S. Lam. Digital Signatures for Flows and Multicasts. *Proceedings of IEEE ICNP '98*, Austin TX, Oct 1998.

Efficient Authentication and Signing of Multicast Streams over Lossy Channels*

Adrian Perrig† Ran Canetti‡ J. D. Tygar† Dawn Song†

†UC Berkeley, ‡IBM T.J. Watson

{perrig,tygar,dawnsong@cs.berkeley.edu, canetti@watson.ibm.com}

Abstract

Multicast stream authentication and signing is an important and challenging problem. Applications include the continuous authentication of radio and TV Internet broadcasts, and authenticated data distribution by satellite. The main challenges are fourfold. First, authenticity must be guaranteed even when only the sender of the data is trusted. Second, the scheme needs to scale to potentially millions of receivers. Third, streamed media distribution can have high packet loss. Finally, the system needs to be efficient to support fast packet rates.

We propose two efficient schemes, TESLA and EMSS, for secure lossy multicast streams. TESLA, short for Timed Efficient Stream Loss-tolerant Authentication, offers sender authentication, strong loss robustness, high scalability, and minimal overhead, at the cost of loose initial time synchronization and slightly delayed authentication. EMSS, short for Efficient Multi-chained Stream Signature, provides nonrepudiation of origin, high loss resistance, and low overhead, at the cost of slightly delayed verification.

*This work began in Summer 1999 when Adrian Perrig and Dawn Song were visiting the IBM T. J. Watson research lab. Initial research on stream authentication was done during Summer 1999 by Ran Canetti, Adrian Perrig, and Dawn Song at IBM. Additional improvements were suggested by J. D. Tygar in Fall 1999 at UC Berkeley. Implementation was done in Fall 1999 by Adrian Perrig at UC Berkeley. The work on stream signatures was done by J. D. Tygar, Adrian Perrig, and Dawn Song at UC Berkeley. Additional work was performed by Ran Canetti in Spring 2000. Ran Canetti is at IBM T. J. Watson Research Center, and Adrian Perrig, Dawn Song, and J. D. Tygar are at the Computer Science Division, UC Berkeley. This research was suported in part by the Defense Advanced Research Projects Agency under DARPA contract N6601-99-28913 (under supervision of the Space and Naval Warfare Systems Center San Diego), by the National Science foundation under grant FD99-79852, and by the United States Postal Service under grant USPS 1025 90-98-C-3513. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government or any of its agencies, DARPA, NSF, USPS, or IBM.

1 Introduction

As the online population continues to expand, the Internet is increasingly used to distribute streamed media, such as streamed radio and video. We expect this trend to continue.

To enable a widespread and trusted streamed media dissemination, one must first provide sufficient security guarantees. A most prominent security risk from a user point of view is data authenticity. The user needs assurance that the data stream originated from the purported sender. Otherwise, a malicious ISP could replace parts of the stream with its own material. For example, an adversary might alter stock quotes that are distributed through IP multicast. In that scenario, the receiver needs strong sender and data authentication.

The problem of continuous stream authentication is solved for the case of one sender and one receiver via standard mechanisms, e.g. [12, 18]. The sender and receiver agree on a secret key which is used in conjunction with a message authenticating code (MAC) to ensure authenticity of each packet. In case of multiple receivers, however, the problem becomes much harder to solve, because a symmetric approach would allow anyone holding a key (that is, any receiver) to forge packets. Alternatively, the sender can use digital signatures to sign every packet with its private key. This solution provides adequate authentication, but digital signatures are prohibitively inefficient.

Real-time data streams are lossy, which makes the security problem even harder. With many receivers, we typically have a high variance among the bandwidth of the receivers, with high packet loss for the receivers with relatively low bandwidth. Nevertheless, we want to assure data authenticity even in the presence of this high packet loss.

A number of schemes for solving this problem (i.e. authenticating the data and sender in a setting where only the sender is trusted) have been suggested in the past few years [7, 13, 28, 31], but none of these schemes is completely sat-

isfactory. We discuss these schemes in section 4.

This paper presents two very different solutions to the problem of authenticating data streams efficiently in a lossy environment. The first solution, called TESLA (for Timed Efficient Stream Loss-tolerant Authentication), uses only symmetric cryptographic primitives such as pseudo-random functions (PRFs) and message authentication codes (MACs), and is based on timed release of keys by the sender. More specifically, the scheme is based on the following idea: The sender commits to a random key k without revealing it and transmits it to the receivers. The sender then attaches a message authenticating code to the next packet P_i and uses the key k as the MAC key. In a later packet P_{i+1}, the sender decommits to k, which allows the receivers to verify the commitment and the MAC of packet P_i. If both verifications are correct, and if it is guaranteed that packet P_{i+1} was not sent before packet P_i was received, then a receiver knows that the packet P_i is authentic. To start this scheme, the sender uses a regular signature scheme to sign the initial commitment. All subsequent packets are authenticated through chaining.

Our first scheme, TESLA, has the following properties:

- *Low computation overhead.* The authentication involves typically only one MAC function and one hash function computation per packet, for both sender and receiver.
- *Low per-packet communication overhead.* Overhead can be as low as 10 bytes per packet.
- *Arbitrary packet loss tolerated.* Every packet which is received in time can be authenticated.
- *Unidirectional data flow.* Data only flows from the sender to the receiver. No acknowledgments or other messages are necessary after connection setup. This implies that the sender's stream authentication overhead is independent on the number of receivers, so our scheme is very scalable.
- *No sender-side buffering.* Every packet is sent as soon as it is ready.
- *High guarantee of authenticity.* The system provides strong authenticity. By strong authenticity we mean that the receiver has a high assurance of authenticity, as long as our timing and cryptographic assumptions are enforced.[1]
- *Freshness of data.* Every receiver knows an upper bound on the propagation time of the packet.

[1]However, the scheme does not provide non-repudiation. That is, the recipient cannot convince a third party that the stream arrived from the claimed source.

The second scheme, called EMSS (for Efficient Multi-chained Stream Signature), is based on signing a small number of special packets in a data stream; each packet is linked to a signed packet via multiple hash chains. This is achieved by appending the hash of each packet (including possible appended hashes of previous packets) to a number of subsequent packets. Appropriate choice of parameters to the scheme guarantees that almost all arriving packets can be authenticated, even over highly lossy channels. The main features of this scheme are:

- It amortizes the cost of a signature operation over multiple packets, typically about one signature operation per 100 to 1000 packets.
- It tolerates high packet loss.
- It has low communication overhead, between 20 to 50 bytes per packet, depending on the requirements.
- It provides non-repudiability of the sender to the transmitted data.

2 TESLA: Timed Efficient Stream Loss-tolerant Authentication

In this section, we describe five schemes for stream authentication. Each scheme builds up on the previous one and improves it to solve its shortcomings. Finally, scheme V, which we call TESLA (short for Timed Efficient Stream Loss-tolerant Authentication), satisfies all the properties we listed in the introduction. The cryptographic primitives used in this section are reviewed in Appendix A, which also contains a sketch of a security analysis for our scheme.

We use the following notation: $\langle x, y \rangle$ denotes the concatenation of x and y, S stands for sender, and R stands for receiver. A stream $\mathcal{S}$ is divided into chunks M_i (which we also call messages), $\mathcal{S} = \langle M_1, M_2, \ldots, M_l \rangle$. Each message M_i is sent in a packet P_i, along with additional authentication information.

2.1 Threat Model and security guarantee

We design our schemes to be secure against a powerful adversary with the following capabilities:

- Full control over the network. The adversary can eavesdrop, capture, drop, resend, delay, and alter packets.
- The adversary has access to a fast network with negligible delay.
- The adversary's computational resources may be very large, but not unbounded. In particular, this means that

the adversary can perform efficient computations, such as computing a reasonable number of pseudo-random function applications and MACs with negligible delay. Nonetheless the adversary cannot invert a pseudorandom function (or distinguish it from a random function) with non-negligible probability.

The security property we guarantee is that the receiver does not accept as authentic any message M_i unless M_i was actually sent by the sender. A scheme that provides this guarantee is called a *secure stream authentication scheme.*

Note that the above security requirements do not include protection against message duplication. Such protection can (and should) be added separately by standard mechanisms, such as nonces or serial numbers. Schemes I-III below do have protection against message duplication. Note also that we do not address denial-of-service attacks.

2.2 Initial synchronization (preliminary discussion)

All five schemes below begin with an initial synchronization protocol where each receiver compares its local time with that of the sender, and registers the difference. We remark that a *rough upper bound* on the clock difference is sufficient. In fact, all that the receiver needs is a value δ such that the sender's clock is no more than δ time-units ahead of the receiver's clock, where δ can be on the order of multiple seconds.[2] In section 2.8 we describe a simple protocol and discuss scalability issues related to the initial synchronization.

A basic assumption that underlies the security of our scheme is that the local internal clocks of the sender and recipient do not drift too much during a session.

2.3 Scheme I: The Basic Scheme

Here is a summary of scheme I: The sender issues a signed commitment to a key which is only known to itself. The sender then uses that key to compute a MAC on a packet P_i, and later discloses the key in packet P_{i+1}, which enables the receiver to verify the commitment and the MAC of packet P_i. If both verifications are successful, packet P_i is authenticated and trusted. The commitment is realized via a pseudorandom function with collision resistance. More details on the requirements on the pseudorandom functions are in appendix A. This protocol is similar to the Guy Fawkes protocol [1].

We now describe the basic scheme in more detail. The scheme is depicted in Figure 1. We assume that the receiver has an authenticated packet $P_{i-1} = \langle D_{i-1}, \text{MAC}(K'_{i-1}, D_{i-1})\rangle$ to start with (where $D_{i-1} = \langle M_{i-1}, F(K_i), K_{i-2}\rangle$). The fields have the following meanings. M_{i-1} is the message contained by the packet, $K'_i = F'(K_i)$ is the secret key used to compute the MAC of the next packet, and $F(K_i)$ commits to the key K_i without revealing it. The functions F and F' are two different pseudo-random functions. Commitment value $F(K_i)$ is important for the authentication of the subsequent packet P_i. To bootstrap this scheme, the first packet needs to be authenticated with a regular digital signature scheme, for example RSA [27].

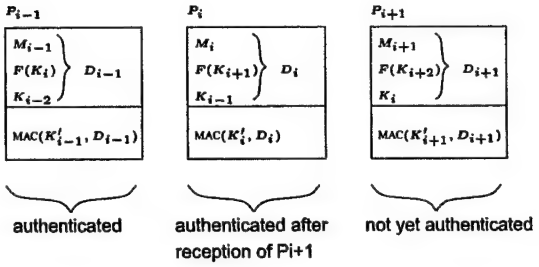


Figure 1. Basic stream authentication scheme. M_i stands for message i, P_i is packet i, K_i denotes the secret key i, F, F' are pseudo-random functions, and $\text{MAC}(K'_i, D_i)$ computes the MAC of packet i using the secret key $K'_i = F'(K_i)$.

To send the message M_i, the sender picks a fresh random key K_{i+1} and constructs the following packet $P_i = \langle D_i, \text{MAC}(K'_i, D_i)\rangle$, where $D_i = \langle M_i, F(K_{i+1}), K_{i-1}\rangle$ and the $\text{MAC}(K'_i, D_i)$ computes a message authenticating code of D_i under key K'_i.

When the receiver receives packet P_i, it cannot verify the MAC instantly, since it does not know K_i and cannot reconstruct K'_i. Packet $P_{i+1} = \langle D_{i+1}, \text{MAC}(K'_{i+1}, D_{i+1})\rangle$ (where $D_{i+1} = \langle M_{i+1}, F(K_{i+2}), K_i\rangle$) discloses K_i and allows the receiver first to verify that K_i is correct ($F(K_i)$ equals the commitment which was sent in packet P_{i-1}); and second to compute $K'_i = F'(K_i)$ and check the authenticity of packet P_i by verifying the MAC of packet P_i.

After the receiver has authenticated P_i, the commitment $F(K_{i+1})$ is also authenticated and the receiver repeats this scheme to authenticate P_{i+1} after P_{i+2} is received.

This scheme can be subverted if an attacker gets packet P_{i+1} before the receiver gets P_i, since the attacker would then know the secret key K_i which is used to compute the MAC of P_i, which allows it to change the message and the commitment in P_i and forge all subsequent traffic. To prevent this attack, the receiver checks the following *security*

[2] Many clock synchronization algorithms exist, for example the work of Mills on NTP [22], and its security analysis [5].

condition on each packet it receives, and drops the packet if the condition does not hold.

> **Security condition**: A data packet P_i arrived *safely*, if the receiver can unambiguously decide, based on its synchronized time and δ_t, that the sender did not yet send out the corresponding key disclosure packet P_j.

This stream authentication scheme is secure as long as the security condition holds. We would like to emphasize that the security of this scheme does not rely on any assumptions on network latency.

In order for the receiver to verify the security condition, the receiver needs to know the precise sending schedule of packets. The easiest way to solve this problem is by using a constant packet rate. The sending time of packet P_i is hence $T_i = T_0 + i/r$ where T_i is the time on the sender's clock and r is the packet rate (number of packets per second). In that case, the security condition which the receiver checks has the following form: $ArrT_i + \delta_t < T_{i+1}$, where $ArrT_i$ stands for the arrival time (on the synchronized receiver's clock) of packet P_i. The main problem with this scheme is that, in order to satisfy the security condition, the sending rate must be slower than the network delay from the sender to the receiver. This is a severe limitation on the throughput of the transmission. In addition, the basic scheme cannot tolerate packet loss. In particular, once a packet is dropped no further packets can be authenticated. We now gradually extend the basic scheme to eliminate these deficiencies.

2.4 Scheme II: Tolerating Packet Loss

To authenticate lossy multimedia streams, tolerating packet loss is paramount. Our solution is to generate a sequence of keys $\{K_i\}$ through a sequence generated through pseudo-random function applications. We denote v consecutive applications of the pseudo-random function F as $F^v(x) = F^{v-1}(F(x))$. By convention, $F^0(x) = x$. The sender picks a random K_n and pre-computes a sequence of n key values, where $K_0 = F^n(K_n)$, and $K_i = F^{n-i}(K_n)$. We call this sequence of values the *key chain*. Each K_i looks pseudorandom to an attacker; in particular, given K_i, the attacker cannot invert F and compute any K_j for $j > i$. On the other hand, the receiver can compute all K_j from a K_i it received, where $j < i$, since $K_j = F^{i-j}(K_i)$. Hence, if a receiver received packet P_i, any subsequently received packet will allow it to compute K_i and $K'_i = F'(K_i)$ and verify the authenticity of P_i. This scheme tolerates an arbitrary number of packet losses.

Similarly, dropping unsafe packets (i.e. those packets where the security condition does not hold) does not cause any problems in the authentication of later packets.

In the basic scheme I, an adversary might try to capture two consecutive packets before the recipient received the first of them, and then forge the packet stream. Although the security condition prevents this, the key chain also prevents this attack, because the initial commitment commits to the entire key chain and it is computationally infeasible for the attacker to invert or find collisions in the pseudo-random function.[3]

An additional benefit is that the key commitment does not need to be embedded in each packet any more. Due to the intractability of inverting the pseudo-random function, any value of the chain is a commitment for the entire chain. Hence the commitment in the initial authenticated packet is sufficient. Figure 2 shows an example of scheme II.

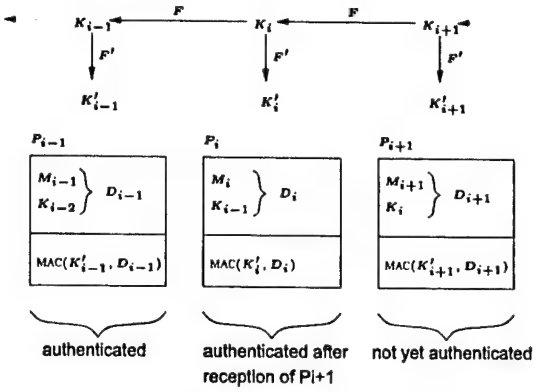


Figure 2. Scheme II. The packet format is the same as in scheme I, except that the commitment $F(K_{i-1})$ is omitted and the keys form a one-way key chain.

2.5 Scheme III: Achieving Fast Transfer Rates

As we mentioned earlier, the receiver needs to be assured that it receives the packet P_i before the corresponding key disclosure packet P_{i+1} is sent by the sender. This condition severely limits the transmission rate of the previous two schemes since P_{i+1} can only be sent after every receiver has received P_i.

We solve this problem by disclosing the key K_i of the data packet P_i in a later packet P_{i+d}, instead of in the following packet, where d is a delay parameter that is set by the sender and announced as the session set-up.

The sender determines the delay d based on the packet rate r, the maximum tolerable synchronization uncertainty

[3] I.e., it is infeasible, given $K_i = F(K_{i+1})$ to find K'_{i+1} such that $F(K'_{i+1}) = K_i$. Even if the attacker could find such a collision $F(K'_{i+1}) = K_i$ then it would be able to forge only a single message M_{i+1}. Forging additional messages would require inverting F, i.e., finding K'_{i+2} such that $F(K'_{i+2}) = K'_{i+1}$.

δ_{tMax}, and the maximum tolerable network delay d_{NMax}. Setting $d = \lceil(\delta_{\text{tMax}} + d_{\text{NMax}})r\rceil$ allows the receiver to successfully verify the security condition even in the case of maximum allowable network delay and maximal synchronization error. The choice of δ_{tMax} and d_{NMax} presents the following tradeoff: Large delay values will cause a large d which results in long delays until the packet authentication. On the other hand, short maximum delays cause the the security condition to drop packets at receivers with a slow network connection. However, multimedia data packets become obsolete if they are received after their segment of the stream was already played or presented to the user. In that case, dropping unsafe packets might not interfere with the multimedia stream since the packets are likely to be obsolete. We stress that the choice of d does not affect the security of the scheme, only its usability.

For the case of a constant packet rate, the security condition is easy to state. We assume that the sending time of the first packet is T_0 and the sending time of packet P_i is $T_i = T_0 + i/r$. To verify the security condition for an incoming packet, the receiver checks that $ArrT_i + \delta_t < T_{i+d}$, where $ArrT_i$ is the arrival time of packet P_i at the receiver.

2.6 Scheme IV: Dealing with Dynamic Packet Rates

Our previous schemes used a fixed or predictable sender schedule, with each recipient knowing the exact sending time of each packet. Since this severely restricts the flexibility of senders, we design a scheme which allows senders to send at dynamic transmission rates, without the requirement that every receiver needs to know about the exact sending schedule of each packet. The solution to this problem is to pick the MAC key and the disclosed key in each packet only on a time interval basis instead of on a packet index basis. The sender uses the same key K_i to compute the MAC for all packets which are sent in the same interval i. All packets sent in interval i disclose the key K_{i-d}.

At session set-up the sender announces values T_0 and T_Δ, where the former is the starting time of the first interval and the latter is the duration of each interval. In addition the delay parameter d is announced. These announcements are signed by the sender. The interval index at any time period t is determined as $i = \lfloor\frac{t-T_0}{T_\Delta}\rfloor$. A key K_i is associated with each interval i. The keys are chained in the same way as in Scheme II. The sender uses the same key $K'_i = F'(K_i)$ to compute the MAC for each packet which is sent in interval i. Every packet also carries the interval index i and discloses the key of a previous interval K_{i-d}. We refer to d as *disclosure lag*. The format of packet P_j is $P_j = \langle M_j, i, K_{i-d}, \text{MAC}(K'_i, M_j)\rangle$. Figure 3 shows an example of this scheme, where $d = 4$.

In this scheme, the receiver verifies the security condition as follows. Each receiver knows the values of T_0, T_Δ, and δ_t. (δ_t is the value obtained from the initial synchronization protocol.) Assume that the receiver gets packet P_j at its local time t_j, and the packet was apparently sent in interval i. The sender can be at most in interval $i' = \lfloor\frac{t_j+\delta_t-T_0}{T_\Delta}\rfloor$. The security condition in this case is simply $i+d > i'$, which ensures that no packet which discloses the value of the key could have been sent yet. Figure 4 illustrates the verification of the security condition.

It remains to describe how the values T_Δ and d are picked. (We stress that the choice of these values does not affect the security of the scheme, only its usability.) Before the sender can pick values for T_Δ and d, it needs to determine the maximum tolerable synchronization uncertainty δ_{tMax}, and the maximum tolerable network delay d_{NMax}. The sender defines $\Delta_{\text{Max}} \stackrel{\text{def}}{=} \delta_{\text{tMax}} + d_{\text{NMax}}$

The sender's choice for T_Δ and Δ_{Max} both present a tradeoff. First, a large value for Δ_{Max} will allow slow receivers to verify the security condition correctly, but requires a long delay for packet authentication. Conversely, a short Δ_{Max} will cause slow receivers to drop packets because the security condition is not satisfied. The second tradeoff is that a long interval duration T_Δ saves on the computation and storage overhead of the key chain, but a short T_Δ more closely achieves the desired Δ_{Max}.

After determining δ_{tMax}, d_{NMax}, and T_Δ, the disclosure lag is $d = \lceil\frac{\delta_{\text{tMax}}+d_{\text{NMax}}}{T_\Delta}\rceil$.

This scheme provides numerous advantages. First, the sender can predict how long a pre-computed key chain lasts, since the number of necessary keys is only time dependent and not on the number of packets sent. Second, the receiver can conveniently verify the security condition and the sender does not need to send its packets at specific intervals (we will discuss the details of this in Section 2.9). Another advantage is that new receivers can easily join the group at any moment. A new group member only needs to synchronize its time with the sender and receive the interval parameters and a commitment to the key chain.

2.7 Scheme V: Accommodate a Broad Spectrum of Receivers

For the previous schemes, we showed that there was a tradeoff in the choice of the key disclosure period. If the time difference is short, the packet can be authenticated quickly, but if the packet travel time is long the security condition will not hold for remote receivers, which forces them to drop the packet. Conversely, a long time period will suit remote receivers, but the authentication time delay may be unacceptable for receivers with fast network access. Since the scheme needs to scale to a large number of receivers and we expect the receivers to have a wide variety of network access, we need to solve this tradeoff. Our approach is to use multiple authentication chains (where each chain is as in scheme IV) with different disclosure periods

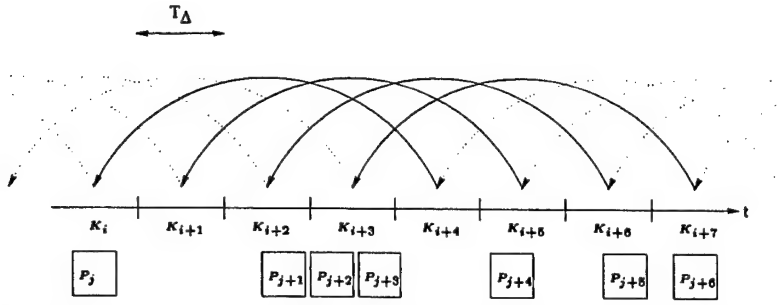


Figure 3. Scheme IV. The MAC key and disclosed key are only dependent on the time interval. The authentication key of P_j is K_i which is disclosed by packets sent during interval $i+4$. In this case, packet P_{j+4} discloses key K_{i+1} which allows the receiver to compute K_i and to authenticate packet P_j. We would like to point out that packets P_{j+2} and P_{j+3} are both authenticated with the same MAC key K'_{i+3}, because they were sent in the same time interval.

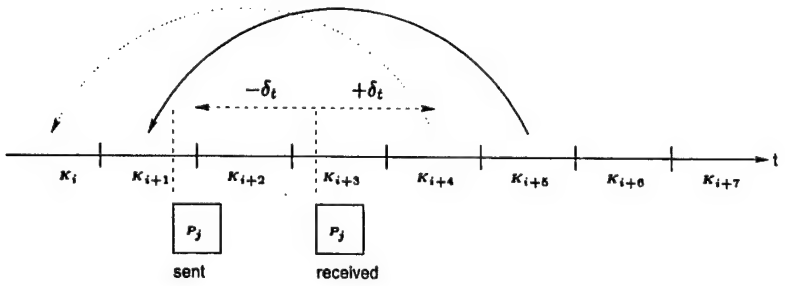


Figure 4. The security condition visualized. The packet P_j is sent in the interval where key K_{i+1} is active. The receiver receives the packet when the sender is in interval $i+3$, but due to the δ_t the sender might already be in interval $i+4$, which discloses key K_i. This is not a problem for the current packet, so key K_{i+1} was not disclosed yet, hence the security condition is satisfied and the packet is safe.

simultaneously. Each receiver can then use the chain with the minimal disclosure delay, sufficient to prevent spurious drops which are caused if the security condition does not hold.

The receiver verifies one security condition for each authentication chain C_i, and drops the packet if none of the conditions are satisfied. Assume that the sender uses n authentication chains, where the first chain has the smallest delay until the disclosure packet is sent, and the nth chain has the longest delay. Furthermore, assume that for the incoming packet P_j, the security conditions for chains C_v $(v < m)$ are not satisfied, and the condition for chain C_m is satisfied. In this case, as long as the key disclosure packets for the chains C_v $(v < m)$ arrive, the receiver's confidence in the authenticity of packet P_j is increasing. As soon as the key disclosure packet for a chain C_v $(v \geq m)$ arrives, the receiver is assured of the authenticity of the packet P_j.

2.8 Initial Synchronization – Further Discussion

Our stream authentication scheme relies on a loose time synchronization between the sender and all the recipients. We call this synchronization loose, because the synchronization error can be large. The only requirement we have is that the client knows an upper bound δ_t on the maximum synchronization error.

Any time synchronization protocol can be used for our scheme, as long as it is robust against an active adversary.

As a proof-of-concept, we present a simple time synchronization protocol which suffices the requirements. The basic protocol is as follows:

$R \rightarrow S$: Nonce

$S \rightarrow R$: {Sender time t_S, Nonce, Interval Rate, Interval Id, Interval start time, Interval key, Disclosure Lag}$_{K_S^{-1}}$

The receiver[4] uses a nonce in its first packet to prevent an attack which replays a previously signed synchronization reply. Besides the current time t_S at the sender, the sender also sends all information necessary to define the intervals and a commitment to the active key chain. The disclosure lag defines the difference in intervals on when the key values are disclosed. Finally, the packet is signed with a regular signature scheme.

For the purposes of our stream authentication scheme, the receiver is only interested in the maximum possible time value at the sender. This simplifies the computation. Figure 5 shows a timing diagram of the synchronization. The receiver sets $\Delta_t = t_S - t_R$ and computes the latest possible sender's time t'_S as follows: $t'_S = t'_R + \Delta_t$, where t'_R is the current receiver's time, and t'_S is the estimated sender time. In the ideal case, the receiver's initial packet arrives at the sender without delay, denoted as time t_1 in the figure. The maximum time discrepancy $\delta_t = \text{RTT}$ (round-trip time).

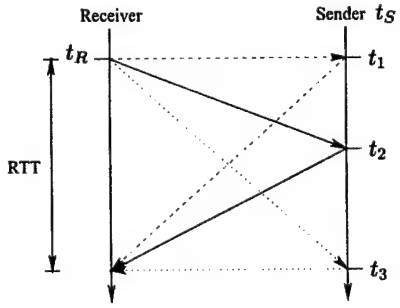


Figure 5. The receiver synchronizes its time with the sender.

Scalability is a major concern for a widely deployed system. If every receiver needs to synchronize its time with the sender, the sender could be a bottleneck. A better solution would use distributed and secure time servers. Initially, the sender synchronizes its time with the time server and computes the maximum synchronization error $\delta_t(S)$. The sender would periodically broadcast the interval information, along with its $\delta_t(S)$ and the current timestamp, digitally signed to ensure authenticity. The receivers can independently synchronize their time to the synchronization server, and individually compute their maximum synchronization error δ_t. Finally, the receivers add up all the δ_t values to verify the security condition. Taking this scheme one step further, we could have a hierarchy of synchronization servers (only the maximum errors need to propagate). We could also imagine synchronizing all the synchronization servers with a satellite signal, for example GPS.

[4]The terms sender and receiver appear reversed in the description of this time synchronization protocol, because we keep their role with respect to the stream authentication scheme. So it is the receiver that synchronizes its time with the sender's.

Combining with multicast group control centers. The general IP multicast model assumes that any host can join the multicast group, receive all group data, and send data to the group [11]. To join the multicast group, the receiver only needs to announce its interest to a local router which takes care of forwarding packets to that receiver. Each joining group member contacts a central server or a group controller to negotiate access rights and session keys. This model is supported by the Secure Multicast Users Group (SMUG) [29] and we adopt it for our secure authentication scheme, which requires that each receiver performs an initial registration (for time synchronization and interval timing information) at the sender or at a central server.

Here is a sketch of a scalable synchronization mechanism that uses this infrastructure: Both senders and receivers synchronize with time synchronization servers which are dispersed in the network. After the synchronization, every entity E knows the time and the maximum error $\delta_t(E)$. The sender S periodically broadcasts a signed message which contains $\delta_t(S)$, along with the interval and key chain commitment information for each authentication chain. A new receiver R therefore only need wait for the broadcast packet allowing it to compute the synchronization error between itself and the sender as $\delta_t = \delta_t(S) + \delta_t(R)$. Based on the δ_t the receiver determines the minimum-delay authentication chain it can use. Hence, the receiver does not need to send any messages to the sender, provided that the sender and receiver have a method to synchronize and the receiver knows the upper bound of the synchronization error δ_t.

Dealing with clock drift. Our authentication protocols assume that there is no clock drift between the sender and the receiver. In practice, however, the software clock can drift (e.g. under heavy load when the timer interrupt does not get serviced). Also, an attacker might be able to change the victim's time (e.g. by sending it spoofed NTP messages). A solution to these problems is that the receiver always consults its internal hardware clock, which has a small drift and which is hard for an attacker to disturb. Furthermore, the longer authentication chains in Scheme V tolerate an authentication delay on the order of tens of seconds, giving us a large security margin. It is reasonable to assume that the hardware clock does not drift tens of seconds within one session. Finally, the receiver can re-synchronize periodically, if the hardware clock appears to drift substantially.

2.9 Implementation Issues

We implemented a TESLA prototype in Java. We use the MD5 hash function [26] in conjunction with the HMAC construction [4] for our pseudo-random function and the MAC. To limit the communication overhead, we only use the 80 most significant bits of the output, which saves space over the standard 96 bits and gives sufficient security. The initial synchronization packet is signed using an 1024 bit RSA signature [27].

In our design, all of the functionality for TESLA remains in the application layer. This design principle follows the approach of ALF, which Tennenhouse and Clark introduce [9]. In ALF, the application knows best how to handle the data, as opposed to placing services in the network or transport layer of the OSI stack. ALF is ideally suited for TESLA. Since the authentication of packets is delayed, the application knows best how to handle unauthenticated information, which might be declared invalid later. We see two main possibilities for the application to interact with a TESLA module on the receiver side. First, we could buffer all incoming packets and deliver them only after their authenticity is assured. Second, we could deliver incoming packets directly, but inform the application through an upcall as soon as a packet is authenticated or if the packet is faulty. We implemented the second alternative.

On the other hand, however, there are also arguments for implementing TESLA in the transport layer, along with other security services [18]. Both variants of interaction with the application are possible. In the first case, the network layer buffers the stream data, and forwards it as soon as the data authenticity is guaranteed.[5] In the second case, the network layer would directly forward the data to the application, but this would require another mechanism for the network layer to inform the application about the validity of the data. To prevent applications from using data that was not authentic, we can imagine a scheme where the sender encrypts the data in each packet with a separate key and releases the key in a later packet. In this case, the application would receive the encrypted data, but could only use it after it receives the decryption key.

We use UDP datagrams for all communication to simulate multicast datagrams. We would like to point out that using a reliable transport protocol such as TCP does not make sense in this setting, because TCP interferes with the timing of packet arrival and does not announce incoming packets to the application if the previous packets did not arrive. This is a problem since our TESLA module resides in the application space. Furthermore, since TESLA is particularly well suited for lossy data streams, UDP makes perfect sense, whereas TCP is used in settings which require reliable communication.

To simplify the exposition of the protocols, we consider the case of Scheme IV, which uses one authentication chain only, as an examplar.

Sender Tasks

The sender first needs to define the following parameters for TESLA:

- The number of authentication chains
- The interval rate for each authentication chain
- The disclosure delay for each authentication chain

The number of authentication chains is dependent on the heterogeneity of network delay across receivers, the delay variance, and the desired authentication delay. For example, if we use TESLA in a LAN setting with a small network delay and low delay variance, the sender can use one single authentication chain with a disclosure lag of about one RTT, which can be as low as a few milliseconds. The other extreme is a radio broadcast over the Internet with millions of receivers. Some receivers will have high-speed network access with a low delay, others use dialup modem lines, and yet others might be connected through a wireless link with considerable delay, which can be on the order of seconds. To accommodate the latter category, which might also have a large synchronization error on the order of seconds, the longest authentication chain needs to have an disclosure delay as long as 15 to 30 seconds. Such a long delay is not acceptable to the high-speed users. A second authentication chain with a small disclosure delay around 1 - 2 seconds is appropriate. To close the wide gap between the high-end and the low-end users, a third chain with a delay of 5 to 10 seconds will appeal to the modem users.

Initially, the sender picks a random key K_n and computes and stores the entire chain of keys $K_i = F(K_{i+1})$.

Receiver Tasks

The receiver initially synchronizes with the sender and determines the accuracy δ_t. The sender also sends all interval information and the disclosure lag to the receiver, which is necessary to verify the security condition. The authenticated synchronization packet also contains a disclosed key value, which is a commitment to the key value chain.

For each incoming packet, the receiver first verifies the security condition. It then checks whether the disclosed key value is correct, which can be verified by applying the HMAC-MD5 (our pseudo-random function) until it can verify equality with a previously authenticated commitment.

[5] The argumentation against this method claims that it would put too much burden on the network layer to buffer data packet. For the case of IP fragmentation, however, the network layer already buffers data and forwards it to the application only when the entire packet is complete.

Block size	16	64	256	1024
MD5	256410	169491	72463	22075
HMAC-MD5	75187	65359	39525	17605

Table 1. Performance of primitives of the Cryptix native Java library. The performance is displayed in the number of operations per second.

Packet size (bytes)	64	256	1024
One authentication chain	27677	23009	8148
Two authentication chains	19394	14566	7402
Three authentication chains	14827	13232	6561
Four authentication chains	12653	11349	5914

Table 2. Performance of our packet authentication scheme for a varying number of authentication chains. All performance numbers are in packets per second.

To minimize the computation overhead, the receiver reconstructs and stores the chain of key values. Since the MAC cannot be verified at this time, the receiver adds the triplet ⟨Packet Hash, Interval, MAC value⟩ to the list of packets to be verified, sorted by interval value. Instead of storing the entire packet, the receiver computes and stores only the hash value of the packet. If the incoming disclosed MAC key was new, the receiver updates the key chain and checks whether it can verify the MAC of any packets on the packet list. In the case a MAC does not verify correctly, the library throws an exception to warn the application. Finally, the packet is delivered to the application.

A possible denial-of-service attack is an attacker sending a packet marked as being from an interval far in the future. A receiver would then spend much time to update its key chain. A simple remedy against this attack would be for the receiver to reject packets if they could not have been sent yet (along the lines of the security condition).

A drawback of this stream authentication scheme is that each receiver needs to store the key chain and packet information to verify the packet authenticity. While the key chain is small (since only a few intervals per seconds are used in practice), the amount of storage required can be large for long authentication delays and fast sender rates. In our implementation, only the 80 bit hash and the interval are stored per packet, which amounts to 12 bytes.

Performance

For each outgoing packet, the sender only needs to compute one HMAC function per packet per authentication chain, since the key chain can be pre-computed. Table 1 shows the performance of the MD5, and HMAC-MD5 functions provided by Cryptix [10] running on a 550 MHz Pentium III Linux PC. The Java code was executed by the JIT compiler which comes with the JDK 1.1.8 provided by IBM [17].

We analyze the performance of our stream authentication scheme by measuring the number of packets per second that a sender can create. Table 2 shows the packet rates for different packet sizes and different numbers of authentication chains. We suspect that an optimized C implementation might be at least twice as fast.

The communication overhead of our prototype is 24 bytes per authentication chain. Since we use 80 bit HMAC-MD5, both the disclosed key and the MAC are 10 bytes long. The remaining four bytes are used to send the interval index.

Also, the overhead of pre-computing the key chain is minimal. In our experiments we use an interval length of 1/10th of a second. To pre-compute a key chain long enough to authenticate packets for one hour, the sender pre-computation time is only $36000/74626 \approx 0.5$ seconds.

The computational overhead on the receiver side is the same as on the sender side, except that the receiver needs to recompute the key chain while the sender can pre-compute it. However, the overhead of computing the key chain is negligible, since it involves computing one HMAC functions in each time interval, and in practice only tens of intervals are used per second.

3 EMSS: Efficient Multi-chained Stream Signature

TESLA does not provide non-repudiation. Most multimedia applications do not need non-repudiation since they discard the data after it is decoded and played. Stream signature schemes are still important, however, for the following two cases. First, some applications really do need continuous non-repudiation of each data packet, but we could not find a compelling example. Second, and more importantly, in settings where time synchronization is difficult, TESLA might not work. We present EMSS (Efficient Multi-chained Stream Signature), to achieve non-repudiation which also achieves sender authentication.

The requirements for our stream signature scheme are as follows:

- Non-repudiation for each individual packet
- Continuous non-repudiation of packets
- Robust against high packet loss

- Low computation and communication overhead
- Real-time stream content
- No buffering of packets at the sender required

3.1 Our Basic Signature Scheme

To achieve non-repudiation, we rely on a conventional signature scheme, for example RSA [27] or Rohatgi's k-times signature scheme [28]. Unfortunately, the computation and communication overhead of current signature schemes is too high to sign every packet individually. To reduce the overhead, one signature needs to be amortized over multiple packets.

Our basic solution bases on the following scheme to achieve non-repudiation of a sequence of packets. Packet P_i includes a hash $H(P_{i-1})$ of the previous packet P_{i-1}. By sending a *signature packet* at the end of the stream, which contains the hash of the final packet along with a signature, we achieve non-repudiation for all packets. To achieve robustness against packet loss, each packet contains multiple hashes of previous packets, and furthermore, the final signature packet signs the hash of multiple packets. Figure 6 shows an example, where each packet contains the hash of the two previous packets, and where the signature packet contains the hash of the last two packet and the signature.

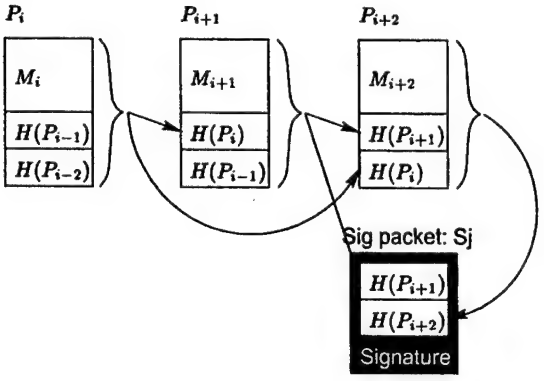


Figure 6. We achieve non-repudiation through periodic signature packets, which contain the hash of several data packets, and the inclusion of the hash of the current packet within future packets. The inclusion of multiple hashes achieves robustness against packet loss.

In order for the sender to continuously verify the signature of the stream, the sender sends periodic signature packets. Since the receiver can only verify the signature of a packet after it receives the next signature packet, it is clear that the receiver experiences a delay until packet verification.

To simplify the following discussion, we describe this scheme as a graph problem and use the corresponding terminology. Namely, we use the term node instead of packet, and edge instead of hash link. We define the length of an edge as $L(E_{ij}) = |i - j|$, where i and j are the id's of the corresponding nodes. If packet P_j contains the hash of packet P_i, we draw a directed edge starting at P_i to P_j. We call P_j a *supporting packet* of P_i. Similarly, an edge points from a packet P_k to a signature packet S_l, if S_l contains the hash of P_k. We assume that some of the packets are dropped between the sender and the receiver. All nodes which correspond to dropped packets are removed from the graph. A packet P_i is *verifiable*, if there exists a path from P_i to any signature packet S_j.

This stream signature scheme has the following parameters:

- Number of edges per node
- Length and distribution of edges
- Frequency of signature nodes
- Number and distribution of incoming edges in signature nodes

These parameters influence the computation and communication overhead, the delay until verification, and the robustness against packet loss. We want to achieve low overhead while retaining high robustness against packet loss and a low verification delay.

To simplify the problem of optimizing all parameters simultaneously, we first focus on the interplay between the number and distribution of edges to achieve high robustness against packet loss. We first consider static edges, which means that all the outgoing and incoming edges of each node have predefined lengths. For example, in a "1-3-7" scheme, the node P_i has outgoing edges to $P_{i+1}, P_{i+3}, P_{i+7}$, and incoming edges from $P_{i-1}, P_{i-3}, P_{i-7}$.

To simplify the problem even further, we initially assume independent packet loss, i.e. each packet has an equal loss probability.[6]

Instead of computing the probability precisely for each node, we wrote a program to perform simulations. We

[6]Our first attempt was to devise an analytical formula to model the probability for each node that it is connected to a signature node. Unfortunately, finding an exact formula is harder than it first appears, so deriving the analytical formula automatically for a given edge distribution remains an open problem. We illustrate this complexity with an example for the recurrence relation which describes the simple 1-2-4 scheme: $P[N-i] = (1-q)\cdot(P[N-i+1]+qP[N-i+2]+(2-q)q^2P[N-i+4]-(1-q)^2q^2P[N-i+5])$, where $P[i]$ is the probability that node i is connected to node N which is signed, and q is the probability that the node is dropped.

checked the accuracy of the simulation program on the cases for which we computed an analytical solution: $1-2-4$ and $1-2-3-4$. Our simulation (with 2500 samples simulating up to 1000 packets before the signature packet) had an absolute error of less than $\pm 2\%$ of the verification probability for these two cases.

We ran extensive simulations to find a good distribution of edges withstanding high amounts of dropped nodes. In our largest simulation, we searched through all combinations of six edges per node, where the maximum length of any edge was 51, and the probability of dropping a node was 60%.[7] In our simulation, we assumed that the final seven nodes all existed and that they all contained an edge to the signature node.

The simulation results were illuminating. The most important finding from the simulation study is that the majority of combinations are robust. Figure 8 illustrates this point. The x-axis ranges over the average probability of verification p. The figure shows how many combinations had that average verification probability p, measured over 400 nodes preceding the signature packet. The figure demonstrates that most of the combinations have high robustness. In fact, 99% of all combinations give an average verification probability over 90%. This finding motivates the use of random edges instead of static edges.

Another interesting result is that the continuous case $1-2-3-4-5-6$ is the weakest combination, and that exponentially increasing edge lengths $1-2-4-8-16-32$ had poor robustness. One of the strongest combinations is $5-11-17-24-36-39$. We show the performance of these three combinations in figure 7. The continuous case has the lowest verification probability, the exponential chain is already much better, and the last case does not seem to weaken as the distance from the signature packet increases.

The assumption of independent packet loss does not hold in the Internet. Many studies show that packet loss is correlated, which means that the probability of loss is much higher if the previous packet is lost. Paxson shows in one of his recent studies that packet loss is correlated and that the length of losses exhibit infinite variance [24]. Borella et al. draw similar conclusions, furthermore they find that the average length of loss bursts is about 7 packets [6].

Yajnik et al. show that a k-state Markov model can model Internet packet loss patterns [32]. For our simulation purposes, the two-state model is sufficient, since it can model simple patterns of bursty loss well [16, 32]. The main advantage of randomizing the edges, however, is visible when we consider correlated packet loss. Figure 9 shows a simulation with 60% packet loss and where the average length of a burst loss is 10 packets. We can clearly see in the figure that the verification probability of the static edge scheme drops exponentially, whereas the random edges still provide a high verification probability.

[7] We chose to use six edges per node, because we wanted to achieve a high average robustness for the case of 60% packet loss and with only five edges did not give us a high verification probability.

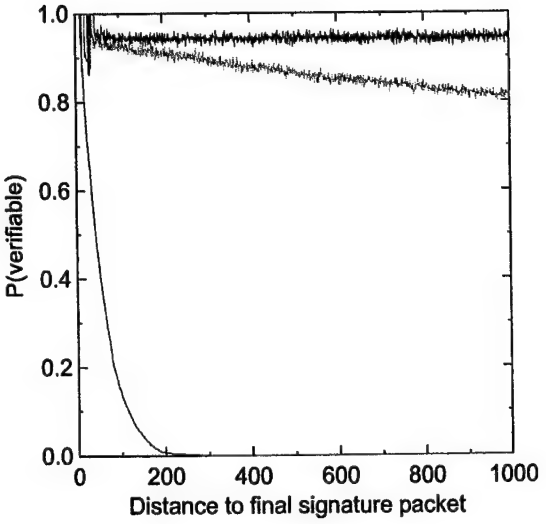


Figure 7. The verification probability for three static cases: Top line: 5-11-17-24-36-39. Middle line: 1-2-4-8-16-32. Bottom line: 1-2-3-4-5-6.

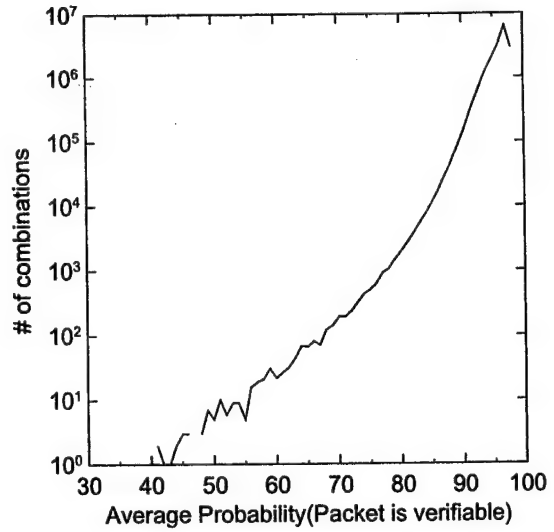


Figure 8. Number of combinations of six hashes that resulted in a given average verification probability. Note that we assume a 60% packet loss probability.

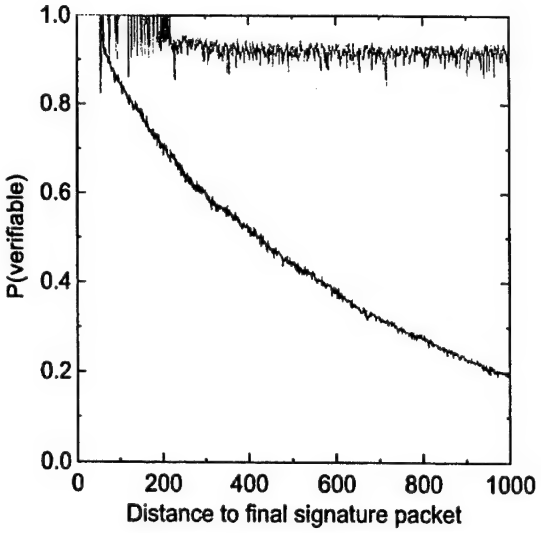


Figure 9. The verification probability for random vs a static case. Top line is random link distribution. Bottom line is 5-11-17-24-36-39.

3.2 The Extended Scheme

The basic scheme has a lot of redundancy. All the supporter packets carry the same hash value of a given packet. In the experiments we use six hashes per packet, hence six packets carry the same hash value. Removing this redundancy might give us a lower communication overhead and improved robustness against loss.

The core idea is to split the hash into k chunks, where a quorum of any k' chunks is sufficient to allow the receiver to validate the information. One approach is to use Rabin's Information Dispersal Algorithm [25], which has precisely this property. Another approach is to produce a hash function with a large number of independent bits, but only look at a limited number of those bits. This can most easily be realized by a family of universal hash functions [8].

The main advantage of this scheme is that *any* k' out of the k packets need to arrive, which has a higher robustness in some circumstances than receiving 1 packet out of d in the basic scheme. For example, if we use the basic scheme with 80-bit hashes and six hashes per packet, the communication overhead is at least 60 bytes, and the probability that at least one out of six packets arrives is $1-q^6$, where q is the loss probability. In contrast, if we use the extended scheme with a hash of 480 bits, chunks of 16 bits, $k = 30$, and $k' = 5$, the probability that the receiver gets more than four packets is $1 - \sum_{i=0}^{i<5} \binom{30}{i} \cdot q^{30-i} \cdot (1-q)^i$. Clearly, the latter probability is much higher. Although both probabilities only provide an upper bound on the verification probability, it still gives an intuition on why the extended scheme provides higher robustness to packet loss.

The simulation confirmed these findings. The extended scheme outperforms the basic scheme in robustness against packet loss. Figure 10 shows a comparison of the two schemes with identical communication overhead.

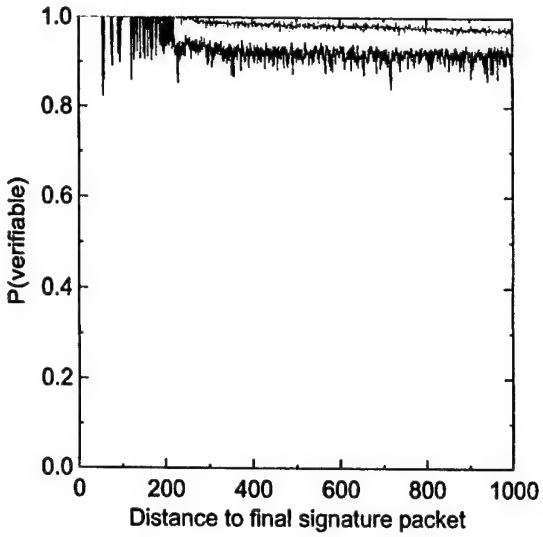


Figure 10. The verification probability for the basic vs. the extended scheme. Top line is the extended scheme. Bottom line is the basic scheme.

3.3 Signature Packets

An important requirement of our scheme signature scheme is that the receiver can continuously verify the signature of packets. Clearly, the receiver can only verify the signature once it can trace the authentication links to a signature packet. Hence, the verification delay depends on the frequency and the transmission reliability of signature packets. The signature packet rate depends on the available computation and communication resources. If we use 1024-bit RSA signatures, a dedicated server can compute on the order of 100 signatures per second. The corresponding communication overhead is 128 bytes for the signature plus 10 bytes for each hash included.

We also performed simulations with signature packets. The parameters included the signature rate, the loss probability of signature packets,[8] and the number of hashes per signature packet. Figure 11 shows the sawtooth-shaped

[8]The loss probability might be different for signature packets if they are sent redundantly or in a higher service class in the context of QoS.

verification probability for a stream with 10% packet loss (bursty loss), the average burst length of dropped packets is 10, the hash is split up into 9 chunks of 27 bits each (spanning a maximum length of 100 packets), hence 3 chunks are necessary to verify a packet, which gives us 81 bits of the signature. The communication overhead per packet is therefore about 35 bytes per packet. The signature packets are sent every 250 packets and they contain 80-bit hashes of 40 packets, and one 1024-bit RSA digital signature which amounts to 128 bytes. Each signature packet is sent twice, so the loss probability of a signature packet is reduced to 1%. The average per-packet overhead in this case is 40 bytes.

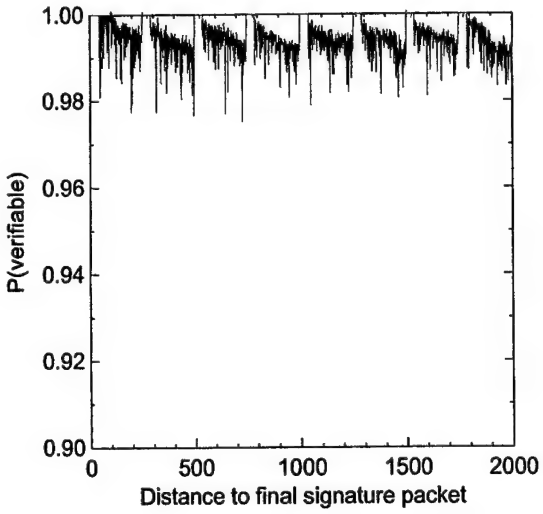


Figure 11. The verification probability for the extended scheme including periodic signature packets.

3.4 Case Study on Two Settings

We consider two different cases of stream distribution and we analyze the overhead of applying EMSS to ensure the non-repudiation of the streamed data.

Case I: Streamed Distribution of Traffic Data

Assume that a municipality has traffic sensors distributed over streets. It broadcasts this data over the Internet so citizens (and robot driven vehicles) can improve their trip planning. The system requirements are as follows:

- The data rate of the stream is about 8 Kbps, about 20 packets of 64 bytes each are sent every second.
- The packet drop rate is at most 5%, where the average length of burst drops is 5 packets.
- The verification delay should be less than 10 seconds.

Many different instantiations of EMSS result in efficient schemes which satisfy these requirements. The following scheme offers low overhead with high verification probability. Each packet has two hashes, and the length of each hash chain element is chosen uniformly distributed over the interval $[1, \ldots, 50]$. Each hash is 80 bits long, hence, only one hash is necessary for verification. A signature packet is sent every 100 packets, or every five seconds, which is not necessary to achieve robustness in this case, but to ensure that the verification delay is less than ten seconds, with high probability. Each signature packet carries the hash of five data packets. The simulation predicts an average verification probability per packet of 98.7%.

The computation overhead is minimal. The sender only needs to compute one signature every five seconds, and only 20 hash functions per second. The communication overhead is low also. Each data packet carries 20 bytes containing the hash of two previous packets.[9] The signature packet contains five hashes and a signature, and its length is hence 50 bytes plus the signature length. Assuming a 1024 bit RSA signature, the signature packet is 178 bytes long. The average per-packet overhead is therefore about 22 bytes, which is much lower than previous schemes, which we review in section 4.

Case II: Real-time Video Broadcast

Assume we want to broadcast signed video on the Internet. The system requirements are as follows:

- The data rate of the stream is about 2 Mbps, about 512 packets of 512 bytes each are sent every second.
- Some clients experience packet drop rates up to 60%, where the average length of burst drops is 10 packets.
- The verification delay should be less than 1 second.

The high packet drop rate makes it difficult for signature packets to reach the receiver. To increase the likelihood of signature packets to arrive, we send them twice — but within a delay, since packet loss is correlated. If we approximate the loss probability by assuming the signature

[9] The packet id's of the packet do not need to be stored in the packet for two reasons. Since the probability of a hash collision is negligible, the receiver can store the hash of the last 50 data packets it received. If any packet contains the same hash value, we consider that packet as verified, if the current packet can be verified. Alternatively, we could build a deterministically computable random graph over the packets, and the receiver would reconstruct it. This alternative would require a packet id in each packet.

packet losses are uncorrelated if they are sent within a delay, the probability that one of them arrives is approximately $1 - 0.6^2 = 0.64$. Since the packet loss is so high and verification delay relatively short, we send a a signature packet every 200 packets. This translates to about 2.5 signatures per second, which we consider as a low computational overhead. We assume that the signature packets have about the same size as the data packets, so in 512 bytes we can fit one 1024-bit RSA signature and the 80 bit hash of 40 previous packets.

We chose these parameters based on good engineering practice. To find better parameters for the number of chunks that the hash is split into and the number of chunks required to verify the packet, we used a simulation. The simulation shows that the best combination for this case uses 50 bytes per packet to insert 25 chunks of two bytes of the hash of previous packets. Including the signature packets, the average communication overhead is about 55 bytes per packet. The simulation predicts the average verification probability over the final 2000 packets of 97%, with the minimum verification probability 90%.

4 Previous Work

We review previous art which deals with the problem of continuous authentication and signature of streams.

Gennaro and Rohatgi introduced techniques for signing digital streams [13]. They present two different schemes, one for the off-line case (the entire stream content is known in advance) and the other for the on-line case (the stream content is generated in real-time). For the off-line case, they suggest signing the first packet and embeding in each packet P_i the hash of the next packet P_{i+1} (including the hash stored in P_{i+1}). While this method is elegant and provides for a stream signature, it does not tolerate packet loss. The biggest disadvantage, however, is that the entire stream of packets needs to be known in advance. The on-line scheme solves this problem through a regular signature of the initial packet and embedding the public key of a one-time signature in each packet, which is used to sign the subsequent packet. The limitation is again that this scheme is not robust against packet loss. In addition, the one-time signature communication overhead is substantial.

Wong and Lam address the problem of data authenticity and integrity for delay-sensitive and lossy multicast flows [31]. They propose to use Merkle's signature trees to sign streams. Their idea to make asymmetric digital signatures more efficient is to amortize one signature generation and verification over multiple messages. Merkle describes how to construct a hash tree over all messages where the signer only digitally signs the root [20, 21]. However, to make this scheme robust against packet loss, every packet needs to contain the signature along with all the nodes necessary to compute the root, which requires large space overhead. In practice, this scheme adds around 200 bytes to each packet (assuming a 1024 bit RSA signature and a signature tree over 16 packets). Another shortcoming is that all messages need to be known to compute the signature tree. This causes delays on the sender side. Furthermore, after the signature computation, all packets are sent at the same time, causing bursty traffic patterns. This burstiness may increase the packet drop rate in the network. Although the computational overhead is amortized over multiple packets, there is still a substantial amount of computation necessary for signature verification, which can consume a substantial amount of resources on low-end receivers (for example battery power). A subtle point is that the per-packet computation increases with the packet loss rate. Since mobile receivers also have less computational power and higher packet loss, the benefit of the amortization is lost. The schemes which we propose in this paper solve these shortcomings.

Rohatgi presents a new scheme which reduces the sender delay for a packet, and which reduces the communication overhead of one-time signatures over previously proposed schemes [28]. He introduces a k-time signature scheme, which is more space efficient than the one-time signatures. Despite all advantages, the scheme still uses 90 bytes for a 6-time public key (which does not include the certificate of the public key) and 300 bytes for each signature. Also, the server requires 350 off-line hash function applications and the client needs 184 hashes on average to verify the signature.

Canetti et al. construct a sender authentication scheme for multicast [7]. Their solution is to use k different keys to authenticate every message with k different MAC's. Every receiver knows m keys and can hence verify m MAC's. The keys are distributed in such a way that no coalition of w receivers can forge a packet for a specific receiver. The communication overhead for this scheme is considerable, since every message carries k MAC's. The server must also compute k MACs before a packet is sent, which makes it more expensive than the scheme we present in this paper. Furthermore, the security of their scheme depends on the assumption that at most a bounded number (which is on the order of k) of receivers collude.

Syverson, Stubblebine, and Goldschlag propose a system which provides asymmetric and unlinkable authentication [30]. In their system, a client proves its right to access the vendor's service through a blinded signature token, which is renewed on each transaction. Through the vendor's blind signature, they achieve unlinkability of transactions. This scheme would not work for stream authentication, because the communication and computation overhead is substantial. Furthermore, the scheme provides unlinkability, which is not needed for authenticating multicast streams.

Anderson et al. [1] present a scheme which provides stream authentication between two parties. Their Guy Fawkes protocol has the following packet format: $P_i = \{M_i, X_i, h(X_{i+1}), h(M_{i+1}, X_{i+1}, h(X_{i+2}))\}$, where M_i denotes message i, X_i stands for a random number, and h is a hash function. Assuming that the receiver received an authentication packet P_i, it can immediately authenticate the following packet P_{i+1}, since P_i contains the commitment $h(M_{i+1}, X_{i+1}, h(X_{i+2}))$ to P_{i+1}. Similarly, P_{i+1} comes with a commitment for P_{i+2}. A drawback of this protocol is that to send message M_i, the following message M_{i+1} needs to be known. Furthermore, this scheme cannot tolerate any packet loss. They propose two methods to guarantee that the keys are not revealed too soon. The first method is that the sender and receiver are in lockstep, i.e. the receiver acknowledges every packet before the sender can send the next packet. This severely limits the transfer time and does not scale to a large number of receivers. The second method to secure their scheme is to time-stamp each packet at a time-stamping service, which introduces additional complexity. The Basic authentication scheme I we propose in this paper is similar to the Guy Fawkes protocol. We improve on Guy Fawkes and construct an efficient stream authentication scheme without these limitations.

We understand that unpublished work by Bob Briscoe at BT research, and Dan Boneh and Philippe Golle, has been proceeding along some similar lines. To the best of our knowledge, all of these groups have been working independently.

5 Acknowledgments

We would like to thank Pankaj Rohatgi for his help during the early stages of the project. We would also like to thank Steve Glassman, Mark Manasse, and Allan Heydon for their helpful comments and discussions. We are also indebted to David Wagner and Bob Briscoe for their relevant feedback and concrete suggestions on how to improve the presentation of this work. Finally, we thank the anonymous reviewers for their helpful suggestions.

References

[1] Ross J. Anderson, Francesco Bergadano, Bruno Crispo, Jong-Hyeon Lee, Charalampos Manifavas, and Roger M. Needham. A new family of authentication protocols. *Operating Systems Review*, 32(4):9–20, October 1998.

[2] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In Yvo Desmedt, editor, *Advances in Cryptology - Crypto '94*, pages 341–358, Berlin, 1994. Springer-Verlag. Lecture Notes in Computer Science Volume 839.

[3] M. Bellare and P. Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In Burt Kaliski, editor, *Advances in Cryptology - Crypto '97*, pages 470–484, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1294.

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Message Authentication using Hash Functions — The HMAC Construction. *RSA Laboratories CryptoBytes*, 2(1), Spring 1996.

[5] Matt Bishop. A Security Analysis of the NTP Protocol Version 2. In *Sixth Annual Computer Security Applications Conference*, November 1990.

[6] M. Borella, D. Swider, S. Uludag, and G. Brewster. Internet packet loss: Measurement and implications for end-to-end qos. In *International Conference on Parallel Processing*, August 1998.

[7] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Infocom '99*, 1999.

[8] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *JCSS No. 18*, (18):143–154, 1979.

[9] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM symposium on Communications architectures and protocols SIGCOMM '90*, pages 200–208, September 26-28 1990.

[10] Cryptix. `http://www.cryptix.org`.

[11] Stephen E. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proceedings of ACM SIGCOMM '88*, August 1988.

[12] T. Dierks and C. Allen. The TLS protocol version 1.0. Internet Request for Comments RFC 2246, January 1999. Proposed standard.

[13] Rosario Gennaro and Pankaj Rohatgi. How to Sign Digital Streams. Technical report, IBM T.J.Watson Research Center, 1997.

[14] Oded Goldreich. Foundations of cryptography (fragments of a book). `http://www.toc.lcs.mit.edu/\~oded/frag.html`, 1998.

[15] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, April 1988.

[16] Mark Handley. Private communication with Adrian Perrig, February 2000.

[17] IBM. Java web page. `http://www.ibm.com/developer/java`.

[18] Ipsec. IP Security Protocol, IETF working group. `http://www.ietf.org/html.charters/ipsec-charter.html`.

[19] Michael George Luby. *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes, 1996.

[20] R. C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology - Crypto '89*, pages 218–238, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 435.

[21] Ralph Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, 1980.

[22] David L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. Internet Request for Comments, March 1992. RFC 1305.

[23] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing (STOC '89)*, 1989.

[24] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[25] M. O. Rabin. The information dispersal algorithm and its applications, 1990.

[26] Ronald L. Rivest. The MD5 message-digest algorithm. Internet Request for Comments, April 1992. RFC 1321.

[27] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[28] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *6th ACM Conference on Computer and Communications Security*, November 1999.

[29] Secure Multicast User Group (SMUG). `http://www.ipmulticast.com/community/smug/`.

[30] Paul F. Syverson, Stuart G. Stubblebine, and David M. Goldschlag. Unlinkable serial transactions. In *Financial Cryptography '97, Springer Verlag, LNCS 1318*, 1997.

[31] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. In *Proc. IEEE ICNP '98*, 1998.

[32] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modelling of the temporal dependence in packet loss. In *IEEE INFOCOM '99*, New York, NY, March 1999.

A Proof of Security

In this appendix, we present a more formal statement of the security assumptions on our cryptographic primitives and sketch the proof of security for one of our stream authentication schemes. First, here are primitives we use in our schemes.

Message Authentication Codes (MACs). A function family $\{f_k\}_{k\in\{0,1\}^\ell}$ (where ℓ is the key length, taken to be the security parameter) is a secure MAC family if any adversary A (whose resources are bounded by a polynomial in ℓ) succeeds in the following game only with negligible probability. A random ℓ-bit key k is chosen; next A can adaptively choose messages $m_1, \ldots, m_n$ and receive the corresponding MAC values $f_k(m_1) \ldots f_k(m_n)$. A succeeds if it manages to forge the MAC, i.e., if it outputs a pair m, t where $m \neq m_1, \ldots, m_n$ and $t = f_k(m)$. See [2] for more details.

Pseudorandom functions (PRFs). A function family $\{f_k\}_{k\in\{0,1\}^\ell}$ (where ℓ is the key length, taken to be the security parameter) is a pseudorandom function family if any adversary A (whose resources are bounded by a polynomial in ℓ) cannot distinguish between a function f_k (where k is chosen randomly and kept secret) and a totally random function only with negligible probability. That is, a function g is chosen to be either f_k for a random ℓ-bit key, or a random function with the same range. Next A gets to ask the value of g on as many points as it likes. Nonetheless A should be unable to tell whether g is random or pseudorandom. see [14, 19] for more details.

The schemes below make use of the following property of pseudorandom functions: as long as the key k is random (or pseudorandom) and remains unknown, the value $k_1 = f_k(x)$ is also pseudorandom for any fixed and known x. (In our schemes we use the arbitrary value $x = 0$.) This allows us to securely iterate; that is, $k_2 = f_{k_1}(x)$ is also pseudorandom, and so on. Furthermore, the value

$k'_1 = f_k(x')$ where $x \neq x'$ is cryptographically independent from k_1 (as long as k remains secret) and can be used as a key for different cryptographic transforms (such as a MAC).

Target collision resistance. A function family $\{f_k\}_{k\in\{0,1\}^\ell}$ (where ℓ is the key length, taken to be the security parameter) is Target Collision Resistant if any adversary A (whose resources are bounded by a polynomial in ℓ) can win in the following game only with negligible probability. First A generates a value v_1 in the common domain of $\{f_k\}$. Next an ℓ-bit key k is randomly chosen and *given to* A. Next A wins if it generates v_2 such that $f_k(v_1) = f_k(v_2)$. Note that target collision resistance implies *2nd pre-image collision resistance.* See more details in [3, 23].

In our scheme we use a PRF family $\{f_k\}$ that also has the following flavor of target collision resistance. First a key k is chosen at random, and the adversary is given $f_k(0)$. Next the adversary is assumed to be unable (except with negligible probability) to find $k' \neq k$ such that $f_{k'}(0) = f_k(0)$.

Since any PRF family is also a secure MAC family, in our schemes we use the same function family for both purposes. Still, for clarity, in the sequel we differentiate between the cryptographic functionality of a PRF and a MAC.[10]

In addition, we use digital signatures (secure against chosen message attacks, see [15]), where the sender holds the signing key and all receivers hold the corresponding public verification key. The way in which the receivers obtain the verification key is left out of scope.

Security Analysis of Scheme III

For brevity, we only sketch a proof of security of one of the TESLA schemes, specifically Scheme III.

Theorem A.1. *Assume that the PRF, the MAC and the signature schemes in use are secure, and that the PRF has the TCR property described in Section A. Then Scheme IV is a secure stream authentication scheme.*

Proof sketch. For simplicity we assume that the MAC and the PRF are realized by the same function family $\{f_k\}$. (In our implementation, f = HMAC.) Assume for contradiction that Scheme III is not a secure stream authentication scheme. This means that there is an adversary A who controls the communication links and manages, with non-negligible probability, to deliver a message m to a receiver R, such that the sender S has not sent m but R accepts m as authentic and coming from S.

We show how to use A to break the security of one of the underlying cryptographic primitives in use. Specifically, we construct a distinguisher D that uses A to break the security of the function family $\{f_k\}$. That is, D gets access to a black-box g and can tell with non-negligible probability if g is a function f_k where k is a random and secret key, or if alternatively g is a totally random function. For this purpose, D can query g on inputs x of its choice and be answered with $g(x)$.

Distinguisher D works by running A, as follows. Essentially, D simulates for A a network with a sender S and a receiver R. That is:

1. D chooses a number $\ell \in \{1..M\}$ at random, where M is the total number of messages to be sent in the stream. (D hopes that A will forge the ℓth message, m_ℓ.)
2. D chooses signing and verification keys for S, and hands the verification key to A.
3. D hands to A the initial message from S. This message is signed using S's signing key, and contains the key K_0, plus the starting time T_0 and the duration d of a time interval. The key K_0 is generated as in the scheme, with the following twist: Recall that in the the scheme $K_0 = F^n(K_n)$ where $F^i(x) = f_{F^{i-1}(x)}(0)$ and K_n is a randomly chosen value (with the appropriate length). Here, $K_0 = F^{\ell-1}(K_{\ell-1})$ where $K_{\ell-1} = g(0)$.
4. For the first $\ell - 1$ messages in the stream D runs the sender's algorithm in Scheme III with no modifications. Whenever a message m_i (with $i < \ell$) is generated, it is handed to A.
5. Message m_ℓ is generated as in Scheme III, with the following exception: In the scheme, the MAC in m_ℓ should equal $f_{K_\ell}(M_\ell, K_{\ell-1})$ where M_ℓ is the actual data in message m_ℓ. Here, D lets the MAC be $g(M_\ell, K_{\ell-1})$.
6. D inspects the messages that A delivers to the receiver R from the moment A receives m_ℓ and until time $T_0 + \ell \cdot d$. (All times are taken locally within D.) If A delivers a message m' that is different than m_ℓ and has a valid MAC with respect to g (i.e., m' is of the form $m' = (M', K', g(M', K'))$) D decides that g was chosen from the pseudorandom family $\{f_k\}$. Otherwise (i.e., A does not successfully forge a message) D decides that g is a random function.

We sketch the argument demonstrating that D succeeds with non-negligible probability. If g is a truly random function then A has only negligible probability to successfully

[10] In fact, we do not need the full security guarantee of a PRF. It suffices to have a (length-doubling) pseudorandom generator with a similar TCR property to the one described above. Nonetheless, for simplicity we describe our schemes as ones using a full-fledged PRF.

forge the ℓth message in the stream. Therefore, if g is random then D makes the wrong decision only with negligible probability.

On the other hand, we have assumed that if the authentication is done using $\{f_k\}$ then A forges some message with non-negligible probability ϵ. It follows that A forges the ℓth message with probability at least ϵ/ℓ. Furthermore, our timing assumption guarantees that A does so prior to time $T_0 + \ell \cdot d$. It follows that if g is taken from $\{f_k\}$ then D makes the right decision with probability at least ϵ/ℓ (which is non-negligible).

We remark that the above argument fails if A hands R a forged initial message from S, or if for some $i < \ell$ adversary A finds a key K_i' that is different than K_i, before time $T_0 + i \cdot d$. However, in these cases the security of the signature scheme or the target collision resistance of $\{f_k\}$ is compromised, respectively. □

Efficient and Secure Source Authentication for Multicast

Adrian Perrig[†*] Ran Canetti[‡] Dawn Song[†] J. D. Tygar[†]

[†]UC Berkeley, *Digital Fountain, [‡]IBM T.J. Watson

{perrig,dawnsong,tygar@cs.berkeley.edu, canetti@watson.ibm.com}

Abstract

One of the main challenges of securing multicast communication is source authentication, or enabling receivers of multicast data to verify that the received data originated with the claimed source and was not modified enroute. The problem becomes more complex in common settings where other receivers of the data are not trusted, and where lost packets are not retransmitted.

Several source authentication schemes for multicast have been suggested in the past, but none of these schemes is satisfactorily efficient in all prominent parameters. We recently proposed a very efficient scheme, TESLA, that is based on initial loose time synchronization between the sender and the receivers, followed by delayed release of keys by the sender.

This paper proposes several substantial modifications and improvements to TESLA. One modification allows receivers to authenticate most packets as soon as they arrive (whereas TESLA requires buffering packets at the receiver side, and provides delayed authentication only). Other modifications improve the scalability of the scheme, reduce the space overhead for multiple instances, increase its resistance to denial-of-service attacks, and more.

1 Introduction

With the growth and commercialization of the Internet, simultaneous transmission of data to multiple receivers becomes a prevalent mode of communication. Often the transmitted data is streamed and has considerable bandwidth. To avoid having to send the data separately to each receiver, several multicast routing protocols have been proposed and deployed, typically in the IP layer. (Examples include [12, 13, 23, 16, 6]). The underlying principle of multicast communication is that each data packet sent from the source reaches a number of receivers.

Securing multicast communication introduces a number of difficulties that are not encountered when trying to secure unicast communication. See [9] for a taxonomy of multicast security concerns and some solutions. A major concern is *source authentication*, or allowing a receiver to ensure that the received data is authentic (i.e., it originates with the source and was not modified on the way), even when none of the other receivers of the data is trusted. Providing source authentication for multicast communication is the focus of this work.

Simply deploying the standard point-to-point authentication mechanism (i.e appending a message authentication code to each packet, computed using a shared key) does not provide source authentication in the case of multicast. The problem is that any receiver that has the shared key can forge data and impersonate the sender. Consequently, it is natural to look for solutions based on asymmetric cryptography to prevent this attack, namely digital signature schemes. Indeed, signing each data packet provides good source authentication; however, it has high overhead, both in terms of time to sign and verify, and in terms of bandwidth. Several schemes were proposed that mitigate this overhead by amortizing a single signature over several packets, e.g. [14, 33, 29]. However, none of these schemes is fully satisfactory in terms of bandwidth and processing time, especially in a setting where the transmission is lossy and some data packets may never arrive. Even though some schemes amortize a digital signature over multiple data packets, a serious denial-of-service attack is usually possible where an attacker floods the receiver with bogus packets supposedly containing a strong signature. Since signature verification is computationally expensive, the receiver is overwhelmed verifying the signatures.

Another approach to providing source authentication uses only symmetric cryptography, more specifically on message authentication codes (MACs), and is based on delayed disclosure of keys by the sender. This technique was first used by Cheung [11] in the context of authenticating communication among routers. It was then used in the Guy Fawkes protocol [1] for interactive unicast communication. In the context of multicast streamed data it

was proposed by several authors [8, 4, 5, 25]. In particular, the TESLA scheme described in [25] was presented to the reliable multicast transport (RMT) working group [26] of the IETF and the secure multicast (SMuG) working group [30] of the IRTF and was favorably received. TESLA is particularly well suited to provide the source authentication functionality for the MESP header [10], or for the ALC protocol proposed by the RMT [19]. Consequently, an Internet-Draft describing the scheme was recently written [24].

The main idea of TESLA, is to have the sender attach to each packet a MAC computed using a key k known only to itself. The receiver buffers the received packet without being able to authenticate it. If the packet is received too late, it is discarded. A short while later, the sender discloses k and the receiver is able to authenticate the packet. Consequently, a single MAC per packet suffices to provide source authentication, provided that the receiver has synchronized its clock with the sender ahead of time.

This idea seems quite attractive at first. However, it has several shortcomings. This work points to these shortcomings and proposes methods to overcome them. Our description is based mostly on TESLA, although the improvements apply to the other schemes as well. We sketch some of these points:

1. In TESLA the receiver has to buffer packets, until the sender discloses the corresponding key, and until the receiver authenticates the packets. This may delay delivering the information to the application, may cause storage problems, and also generates vulnerability to denial-of-service (DoS) attacks on the receiver (by flooding it with bogus packets). We propose a method that allows receivers to authenticate most packets immediately upon arrival, thus reducing the need for buffering at the receiver side and in particular reduces the susceptibility to this type of DoS attacks.

   This improvement comes at the price of one extra hash per packet, plus some buffering at the sender side. We believe that buffering at the sender side is often more reasonable and acceptable than buffering at the receiver side. In particular, it is not susceptible to this type of DoS attacks.

   We also propose other methods to alleviate this type of DoS attacks. These methods work even when the receiver buffers packets as in TESLA.

2. When operating in an environment with heterogenous network delay times for different receivers, TESLA authenticates each packet using multiple keys, where the different keys have different disclosure delay times. This results in larger overhead, both in processing time and in bandwidth. We propose a method for achieving the same functionality (i.e., different receivers can authenticate the packets at different delays) with a more moderate increase in the overhead per packet.

3. In TESLA the sender needs to perform authenticated time synchronization individually with each receiver. This may not scale well, especially in cases where many receivers wish to join the multicast group and synchronize with the sender at the same time. This is so, since each synchronization involves a costly public-key operation. We propose a method that uses only a single public-key operation per time-unit, regardless of the number of time synchronizations performed during this time unit. This reduces the cost of synchronizing with a receiver to practically the cost of setting up a simple, unauthenticated connection.

4. We also explore time synchronization issues in greater depth and describe direct and indirect time synchronization. For the former method, the receiver synchronizes its time directly with the sender, in the latter method both the sender and receiver synchronize their time with a time synchronization server.

   For both cases, we give a detailed analysis on how to choose the key disclosure delay, a crucial parameter for TESLA.

5. TESLA assumes that all members have joined the group and have synchronized with the sender before any transmission starts. In reality, receivers may wish to join after the transmission has started; furthermore, receivers may wish to receive the transmission immediately, and perform the time synchronization only later. We propose methods that enable both functionalities. That is, our methods allow a receiver to join in "on the fly" to an ongoing session; they also allow receivers to synchronize at a later time and authenticate packets only then.

Organization Section 2 reviews TESLA, providing further details than in [25]. Section 3 contains the improvements and extensions proposed in this paper. Section 4 provides further discussion on the security of the improved scheme, with emphasis on resistance to denial-of-service attacks.

2 An Overview of TESLA

The security property TESLA guarantees is that the receiver never accepts M_i as an authentic message unless M_i was actually sent by the sender. Note that TESLA does not provide non-repudiation, that is, the receiver cannot convince a third party that the stream arrived from the claimed source.

TESLA is efficient and has a low space overhead mainly because it is based on symmetric-key cryptography. Since source authentication is an inherently asymmetric property (all the receivers can verify the authenticity but they cannot produce an authentic data packet), we use a delayed disclosure of keys to achieve this property. Similarly, the data authentication is delayed as well. In practice, the authentication delay is on the order of one round-trip-time (RTT).

TESLA has the following properties. First, it has a low computation overhead, which is typically only one MAC function computation per packet, for both sender and receiver. TESLA also has a low per-packet communication overhead, which is about 20 bytes per packet. In addition, TESLA tolerates arbitrary packet loss. Each packet that is received in time can be authenticated. Except for an initial time synchronization, it has only unidirectional data flow from the sender to the receiver. No acknowledgments or other messages are necessary. This implies that the sender's stream authentication overhead is independent of the number of receivers, hence TESLA is very scalable. TESLA can be used both in the network layer or in the application layer. The delayed authentication, however, requires buffering of packets until authentication is completed.

For TESLA to be secure, the sender and the receiver need to be loosely time synchronized, which means that the synchronization does not need to be precise, but the receiver needs to know an upper bound on the sender's time.

2.1 Sender Setup

In our model, a sender distributes a stream of data composed of message chunks $\{M_i\}$. Generally, the sender sends each message chunk M_i in one network packet P_i. Many multicast distribution protocols do not retransmit lost packets. The goal is therefore that the receiver can authenticate each message chunk M_i separately.

For the purpose of TESLA, the sender splits the time into even intervals I_i. We denote the duration of each time interval with T_{int}, and the starting time of the interval I_i is T_i. Trivially, we have $T_i = T_0 + i * T_{int}$. In each interval, the sender may send zero or multiple packets.

Before sending the first message, the sender determines the sending duration (possibly infinite), the interval duration, and the number N of keys of the key chain. This key chain is analogous to the one-way chain introduced by Lamport [18], and the S/KEY authentication scheme [15]. The sender picks the last key K_N of the key chain randomly and pre-computes the entire key chain using a pseudo-random function F, which is by definition a one-way function. Each element of the chain is defined as $K_i = F(K_{i+1})$. Each key can be derived from K_N as $K_i = F^{N-i}(K_N)$, where $F^j(k) = F^{j-1}(F(k))$ and $F^0(k) = k$. Each key of the key chain corresponds to one interval, i.e., K_j is active in interval I_j.

Since we do not want to use the same key multiple times in different cryptographic operations, we use a second pseudo-random function F' to derive the key which is used to compute the MAC of messages in each interval (we will explain the algorithm in detail later). Hence, $K'_i = F'(K_i)$. Figure 1 depicts this key derivation. We propose to use HMAC in conjunction with a cryptographically secure hash function for the pseudo-random function [2]. For example, a possibility is to use the following: $F(x) = \text{HMAC}(x, 0)$ and $F'(x) = \text{HMAC}(x, 1)$, where 0 and 1 are 8-bit integers. Note that the first argument of the MAC function is the key and the second argument is the data.

2.2 Bootstrapping a new Receiver

TESLA requires an initially authenticated data packet to bootstrap a new receiver. This authentication is achieved with a digital signature scheme, such as RSA [28], or DSA [32].

We consider two options for synchronizing the time, direct and indirect synchronization. We improve the time synchronization from our original work and describe the details in section 3.3. Whichever time synchronization mechanism is used, the receiver only needs to know an upper bound on the sender time.

The initial authenticated packet contains the following information about the time intervals and key chain:

- The beginning time of a specific interval T_j, along with its id I_j
- The interval duration T_{int}
- Key disclosure delay d (unit is interval)
- A commitment to the key chain K_i ($i < j - d$ where j is the current interval index)

2.3 Sending Authenticated Packets

Each key of the key chain is used in one time interval. However many messages are sent in each interval, the key which corresponds to that interval is used to compute the MAC of all those messages. This allows the sender to send packets at any rate and to adapt the sending rate dynamically. The key remains secret for d-1 future intervals. Packets sent in interval I_j can hence disclose key K_{j-d}. As soon as the receivers receive that key, they can verify the authenticity of the packets sent in interval I_{j-d}.

The construction of packet P_j sent in interval I_i is: $\{M_j \mid \text{MAC}(K'_i, M_j) \mid K_{i-d}\}$.

Figure 1 shows the key chain construction and the MAC key derivation. If the disclosure delay is 2 intervals, the packet P_{j+4} sent in interval I_{i+2} discloses key K_i. From this key, the receiver can also recover K_{i-1} and verify the MAC of P_j, in case P_{j+3} is lost.

2.4 Receiver Tasks

Since the security of TESLA depends on keys that remain secret until a pre-determined time period, the receiver must verify for each packet that the key, which is used to compute the MAC of that packet, is not yet disclosed by the sender. Otherwise, an attacker could have changed the message data and re-computed the MAC. This motivates the security condition, which the receiver must verify for each packet it receives.

Security condition: A packet arrived *safely*, if the receiver is assured that the sender cannot yet be in the time interval in which the corresponding key is disclosed.

The intuition is that if a packet satisfies the security condition, then no attacker could have altered it in transit, because the corresponding MAC key is not yet disclosed. In case the security condition is not valid, the receiver must drop that packet, because the authenticity is not assured any more. We would like to emphasize that the security of this scheme does not rely on any assumptions on network propagation delay. The original paper sketches a security proof [25].

We now explain how the authentication with TESLA works with a concrete example. When the receiver receives packet P_j sent in interval I_i at local time t_c, it computes an upper bound on the sender's clock t_j (we describe in section 3.3 how to compute this). To evaluate the security condition, the receiver computes the highest interval x the sender could possibly be in, which is $x = \lfloor (t_j - T_0)/T_{int} \rfloor$. The receiver now verifies that $x < I_i + d$ (where I_i is the interval index), which means that the sender must not have been in the interval in which the key K_i is disclosed, hence no attacker can possibly know that key and spoof the message contents.

The receiver cannot, however, verify the authenticity of the message yet. Instead, it stores the triplet $(I_i, M_j, \text{MAC}(K'_i, M_j))$ to verify the authenticity later when it knows K'_i. Two possibilities exist on how to handle the unauthenticated message chunk M_j. The first possibility is to hand M_j to the application, and notify it through a callback mechanism as soon as M_j is verified. The second possibility is to buffer M_j until the authenticity can be checked and pass it to the application as soon as M_j is authenticated.

If the packet contains a disclosed key K_{i-d}, regardless of whether the security condition is verified or not, the receiver checks whether it can use K_{i-d} to authenticate previous packets. Clearly, if it has received K_{i-d} previously, it does not have any work to do. Otherwise, let us assume that the last key value in the reconstructed key chain is K_v. The receiver verifies if K_{i-d} is legitimate by verifying that $K_v = F^{i-d-v}(K_{i-d})$. If that condition is correct, the receiver updates the key chain. For each new key K_w, it computes $K'_w = F'(K_w)$ which might allow it to verify the authenticity of previously received packets.

It is clear that this system can tolerate arbitrary packet loss, because the receiver can verify the authenticity of all received packets that satisfy the security condition eventually.

3 Our Extensions

We extend TESLA in a number of ways to make it more efficient and practical. First, we present a new method to support *immediate authentication*, meaning that the receiver can authenticate packets as soon as they arrive.

Second, we propose optimizations concerning key chains. In particular, for applications that use multiple authentication chains with different disclosure delays, we present a new algorithm that reduces the communication overhead.

Finally, we give discussions on the time synchronization issues and derive a tight lower bound on the key disclosure delay, which makes the scheme much more practical. Next, we remove a scalability limitation of the simple time synchronization protocol. Furthermore, we discuss how a receiver can authenticate received packets even if it is not time synchronized at the moment in which it receives the packet.

3.1 Immediate Authentication

A drawback of the original TESLA protocol is that the receiver needs to buffer packets during one disclosure delay before it can authenticate them. This might not be practical for certain applications if the receivers cannot afford much buffer space and bursty traffic might cause the receivers to drop packets due to insufficient buffer space. Moreover, as we show later in section 4.2, the requirement of receiver buffering introduces a vulnerability to a denial-of-service attack. To solve these problems caused by receiver-buffering, we propose a new method to support *immediate authentication*, which allows the receiver to authenticate packets as soon as they arrive.

The basic observation of this method is that we can replace receiver buffering with sender buffering. If the sender can buffer packets during one disclosure delay, then it could store the hash value of the data of a later packet in an earlier packet and hence as soon as the earlier packet is authenticated, the data in the later packet is authenticated through the hash value as well.

In the new scheme, the sender buffers packets for the

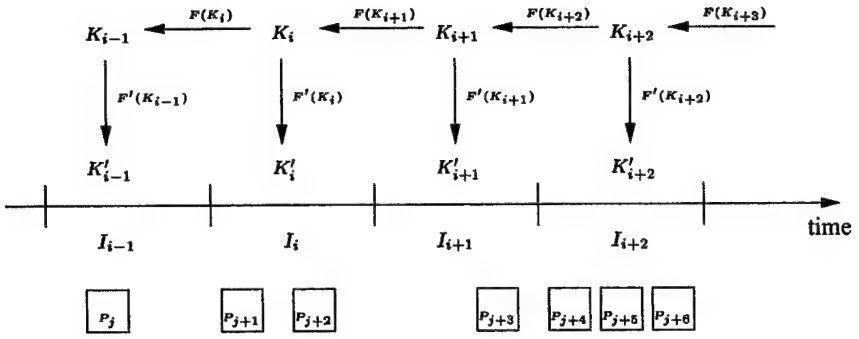


Figure 1: TESLA key chain and the derived MAC keys

duration of one disclosure delay. For simplicity of illustration, we assume that the sender sends out a constant number v of packets per time interval. To construct the packet for the message chunk M_j in time interval T_i, the sender appends the hash value of the message chunk M_{j+vd} to M_j and then computes the MAC value also over $H(M_{j+vd})$ with the key K_i. Figure 2 illustrates how the packet P_j is constructed by appending $H(M_{j+vd})$, $\text{MAC}(K_i, M_j \mid H(M_{j+vd}))$, along with the disclosed key K_{i-d}. (Note that the $\mid$ stands for message concatenation). When the packet P_{j+vd} arrives at the receiver which discloses the key K_i it allows authentication of packet P_j sent in interval I_i. P_j carries a hash of the data M_{j+vd} in P_{j+vd}. If P_j is authentic, $H(M_{j+vd})$ is also authentic and therefore the data M_{j+vd} is immediately authenticated. Also note that if P_j is lost or dropped due to violation of the security condition, P_{j+vd} will not be immediately authenticated and can still be authenticated later using the MAC value.

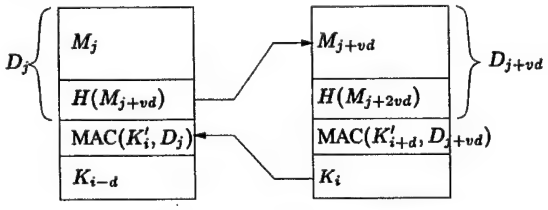


Figure 2: Immediate authentication packet example. $D_j = H(M_{j+vd}) \mid M_j$ and $D_{j+vd} = H(M_{j+2vd}) \mid M_{j+vd}$.

If each packet can only carry the hash of one other packet, it is clear that the sending rate needs to remain constant. Also it is clear that if a packet is lost, the corresponding packet cannot be immediately authenticated. To achieve flexibility for dynamic sending rate and robustness to packet loss, the sender can add the hash values of multiple future packets to a packet, similar to the EMSS scheme [25].

3.2 Concurrent TESLA instances

In this section, we present a space optimization technique in the case the sender uses multiple TESLA instances for one stream.

Choosing the disclosure delay involves a tradeoff. Receivers with a low network delay welcome short key disclosure delays because that translates into a short authentication delay. Unfortunately, receivers with a long network delay could not operate with a short disclosure delay because most of the packets will violate the security condition and hence cannot be authenticated. Conversely, a long disclosure delay would suit the long delay receivers, but causes unnecessarily long authentication delay for the receivers with short network delay. The solution is to use multiple instances of TESLA with different disclosure delays simultaneously, and each receiver can decide which disclosure delay, and hence, which instance to use. A simple approach to use concurrent TESLA instances is to treat each TESLA instance independently, with one key chain per instance. The problem for this approach is that each extra TESLA instance also causes extra space overhead in each packet. If each instance requires 20 bytes per packet (80 bit for key disclosure and 80 bit for the MAC value), using three instances results in 60 bytes space overhead per packet. We present a new optimization which reduces the space overhead of concurrent instances.

The main idea is that instead of using one independent key chain per TESLA instance, we could use the same key chain but a different key schedule for all instances. The basic scheme works as follows. All TESLA instances for a stream share the same time interval duration and the same key chain. Each key K_i in the key chain is associated with the corresponding time interval T_i, and K_i will

be disclosed in T_i.[1] Assume that the sender uses w instances of TESLA, which we denote with $\tau_1 \ldots \tau_w$. Each TESLA instance τ_u has a different disclosure delay d_u, and it will have a MAC key schedule derived from the key schedule shifted by d_u time intervals from the key disclosure schedule. Let $K^u_{i+d_u}$ denote the MAC key used by instance u in time interval T_i. We derive $K^u_{i+d_u}$ as $K^u_{i+d_u} = \text{HMAC}(K_{i+d_u}, u)$. Note that we use HMAC as a pseudo-random function, which is the same key derivation construction as we use in TESLA (see section 2.1 and figure 1). In fact, the keys of the first instance are derived with the same pseudo-random function as the TESLA protocol that uses only one instance. The reason for generating all different, independent keys for each instance is to prevent an attack where an attacker moves the MAC value of an instance to another instance, which might allow it to claim that data was sent in a different interval. Our approach of generating independent keys prevents this attack. Thus to compute the MAC value in packet P_j in time interval T_i, the sender computes one MAC value of the message chunk M_j per instance and append the MAC values to M_j. In particular, for the instance τ_u with disclosure delay d_u, the sender will now use the key $K^u_{i+d_u}$ as mentioned above for the MAC computation.

Figure 3 shows an example with two TESLA instances, one with a key disclosure time of two intervals and the other of four intervals. The lowest line of keys shows the key disclosure schedule, i.e. which key is disclosed in which time interval. The middle and top line of keys shows the key schedule of the first and second instance respectively, i.e. which key is used to compute the MAC for the packets in the given time interval for the given instance. Using this technique, the sender will only need to disclose one key chain no matter how many instances are used concurrently. If each disclosed key is 10 bytes long, then for a stream with m concurrent instances, this technique will save $10(m-1)$ bytes per packet, which is a drastic saving in particular for small packets.

3.3 Time Synchronization Issues

Loose time synchronization is an important component in TESLA. Although sophisticated time synchronization protocols exist, they usually require considerable management overhead. Furthermore, they generally have a high complexity and achieve properties that TESLA does not require. An example is the network time protocol (NTP) by Mills [21]. Bishop performs a detailed security analysis of NTP [7]. For these reasons, we outline a simple and secure time synchronization protocol that suffices the humble requirements of TESLA.

[1] Note that this key schedule is different from the previous schedule described in section 2.1, where key K_i was used to compute the MAC in interval I_i and was disclosed in interval I_{i+d}.

The time synchronization requirement that secures TESLA against an active attacker is that the receiver knows an upper bound of the difference between the sender's local time and the receiver's local time, Δ. For simplicity, we assume the clock drift of both sender and receiver are negligible, otherwise they will simply resynchronize periodically. We denote the real difference between the sender and the receiver's time with δ. Hence for loose synchronization, the receiver does not need know δ but only some Δ that is guaranteed to be greater or equal to δ. To compute Δ, we can use either a direct or an indirect time synchronization method. In the following, we first discuss a simple protocol for direct time synchronization, and next we discuss how to do indirect time synchronization.

Direct Time Synchronization

In direct time synchronization, the receiver performs an explicit time synchronization with the sender. This approach has the advantage that no extra infrastructure is needed to perform the time synchronization. We design a simple two-phase protocol that satisfies the TESLA requirements.

In the protocol, the receiver first records its local sending time t_R and sends a time synchronization request containing a nonce to the sender. Upon receiving the time synchronization request, the sender records its local receiving time t_S and sends the receiver a signed response packet containing t_S and the nonce.

$$R \rightarrow S : \text{Nonce}$$
$$S \rightarrow R : \{\text{Sender time } t_S, \text{Nonce}\}_{K_S^{-1}}$$

Figure 4 shows a sample time synchronization between the receiver and the sender. Upon receiving the signed response, the receiver checks the validity of the signature and the matching of the nonce and computes $\Delta = t_S - t_R$. It is easy to see that the Δ computed this way satisfies the requirement that $\Delta \geq \delta$. Because $\Delta = t_S - t_R = (t_S - t_3) + (t_3 - t_R)$, $t_S - t_3 = \delta$, and $t_3 - t_R$ is the network delay for sending the request from the receiver to the sender which is greater or equal to 0, hence $\Delta \geq \delta$. An interesting point is that the network delay of the response packet and the delay caused by the computation of the digital signature do not influence Δ at all. Since only the initial timestamp matters, it is important that the sender immediately stores the arrival time t_S of the time synchronization request packet. The subsequent processing and propagation delay does not matter.

Because the digital signature operation is computationally expensive, we need to be careful about denial-of-service attacks where an attacker floods the sender with

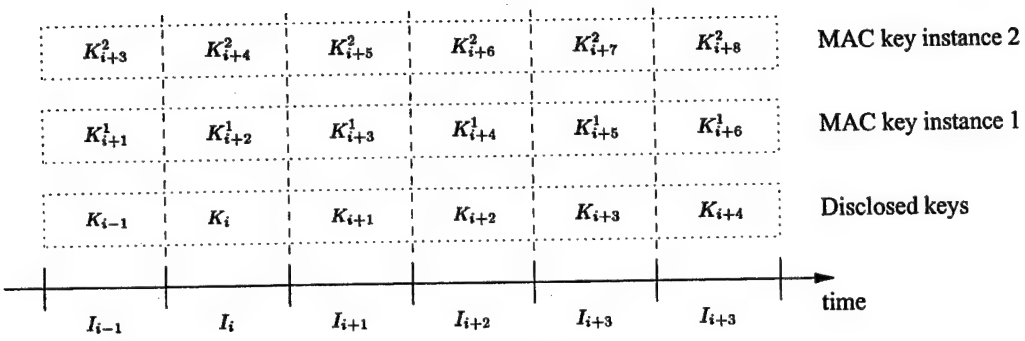


Figure 3: Multiple TESLA instances key chain optimization.

time synchronization requests. Section 4.1 addresses this issue.

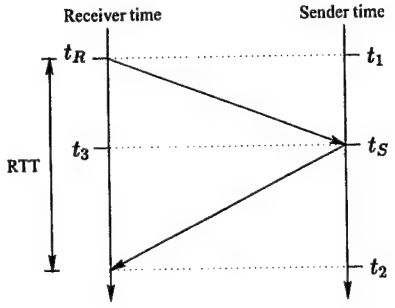


Figure 4: The receiver synchronizes its time with the sender.

Indirect Time Synchronization

In indirect time synchronization, both the sender and the receivers synchronize their time with a time reference and hence the sender and the receiver can reach implicit time synchronization. This approach is favorable especially in cases where the application needs time synchronization with a time reference anyhow. Let $\Delta_{SC} + |\epsilon_{SC}|$ denote the measured upper bound of the difference of the sender's time and the time reference's time with $|\epsilon_{SC}|$ as the maximum error, and let $\Delta_{CR} + |\epsilon_{CR}|$ denote the measured upper bound of the difference of the time reference's time and the receiver's time with $|\epsilon_{CR}|$ as the maximum error. Thus the receiver could reach an implicit time synchronization with the sender as $\Delta = \Delta_{SC} + \Delta_{CR} + |\epsilon_{SC}| + |\epsilon_{CR}|$ with $\epsilon = |\epsilon_{SC}| + |\epsilon_{CR}|$ as the maximum error.

In settings where the receiver is already time synchronized with the time reference, the receiver does not need to send any information to the sender. The sender just needs to periodically broadcast digitally signed packets that contain its time synchronization with the time reference, the time interval and key chain information outlined in section 2.2, along with the sender's maximum synchronization error ϵ_{SC}. A new receiver can start authenticating the data stream right after it receives one of the signed advertisements. This is particularly useful in the case of satellite broadcast.

Delayed Time Synchronization

Another interesting relaxation of the time synchronization requirement is that, if we assume that the receiver's clock drift is negligible during a period of time, then the receiver can receive the data stream from the sender before doing a time synchronization and authenticate the data later after a time synchronization. The receiver only needs to store the arrival time of each packet, so that it can evaluate the security condition after it performed the time synchronization. This is highly useful for many applications, for example a router can use TESLA to authenticate itrace messages [3], and the victim can authenticate the routers' IP markings afterwards when it wants to trace an attacker by performing an approximate time synchronization with the router [31].

3.4 Determining the Key Disclosure Delay

An important parameter to determine for TESLA is the key disclosure delay d. A short disclosure delay will cause packets to violate the security condition and cause packet drop, while a long disclosure delay causes a long authentication delay. Note that although the choice of the disclosure delay does not affect the security of the system, it is an important performance factor. We describe a new method on how to choose a good disclosure delay d. In particular, we show as follows that if RTT is a reasonable upper bound on the round trip time between the receiver and the sender, then in case of using direct time synchronization, we can choose $d = \lceil RTT/T_{int} \rceil + 1$, where T_{int}

is the interval duration. In case of indirect time synchronization, we can choose $d = \lceil (D_{SR}+\epsilon)/T_{int} \rceil + 1$, where ϵ is the sum of both the sender and receiver time synchronization error, and D_{SR} is a reasonable upper bound on the network delay of a packet traveling from the sender to the receiver.

Consider a packet P_i that is constructed using the MAC key K'_j in time interval I_j which will be disclosed d time intervals later. The packet P_i arrives at the receiver at its local time t_i^R. Hence the security condition is that

$$\lfloor \frac{t_i^R + \Delta - T_0}{T_{int}} \rfloor - I_j < d, \qquad (1)$$

where T_0 is the beginning time of the 0th time interval and T_{int} is the time interval duration. Assume packet P_i was sent at the sender's local time t_i^S. Hence $t_i^S < T_j + T_{int} = I_j \cdot T_{int} + T_0 + T_{int}$. We denote the average network delay time from the sender to the receiver with D_{SR} and the average network delay time from the receiver to the sender is D_{RS}, and hence $RTT = D_{RS} + D_{SR}$.

In case of a direct time synchronization, using the same notation as in section 3.3, $\Delta = \delta + (t_3 - t_R) \doteq \delta + D_{RS}, t_i^R + \delta - t_i^S \doteq D_{SR}$, and hence we can derive at the end that a tight bound for d to satisfy the equation 1 is $d = \lceil RTT/T_{int} \rceil + 1$, which allows most of packets to satisfy the security condition and still the receiver would not need to wait much extra longer than necessary to authenticate the packets. Similarly in case of an indirect time synchronization, we can derive that a good d is $d = \lceil (D_{SR} + \epsilon)/T_{int} \rceil + 1$.

4 Security Discussion and Robustness to DoS

Our original paper did not address denial-of-service (DoS) attacks on TESLA. In an IP multicast environment, however, DoS is a considerable threat and requires careful consideration. We discuss potential security problems in this section and show how to strengthen TESLA to thwart them. In particular, we show that there is no DoS attack on the sender if the receivers perform indirect time synchronization. In case of direct time synchronization, we show how to mitigate DoS attacks on the sender. Although there are some potential DoS attacks on the receiver side, we show that TESLA does not add any additional vulnerability to DoS attacks if the receiver has a reasonable amount of buffer space, otherwise we describe schemes that alleviate the exposure to DoS.

4.1 DoS Attack on the Sender

A DoS attack on the sender is not possible if TESLA is used with indirect time synchronization, because the sender does not keep per-receiver state or perform per-receiver operations. In the case of direct time synchronization, a DoS attack is possible, since the sender is required to digitally sign each nonce included in a time synchronization request. An attacker can perform a DoS by flooding the sender with requests.

This response packet needs to be authenticated with a digital signature scheme, such as RSA [28], or DSA [32]. Since public-key signature algorithms are computationally expensive, the signing of the response packet can become a performance bottleneck for the sender. A simple trick can alleviate this situation. The sender can aggregate multiple requests, compute and sign a Merkle hash tree that is generated from all the requester's nonces [20]. Figure 5 shows how such a hash tree is constructed. If N_h is the root of the hash tree, N_h would be included in the signed part of the response packet instead of the receiver's nonce N_r. To verify the digital signature of the response packet, each receiver would reconstruct the hash tree. Since it does not know the other receiver's nonces that are part of the hash tree, the sender would include the nodes of the tree necessary to reconstruct the root node. For the example in figure 5, the packet returned to receiver A would include N_b and H_{cd}. Receiver A can reconstruct the root node H_{ad} from these values and its own nonce N_a as follows: $H_{ad} = H(H(N_a, N_b), H_{cd})$. Note that the number of nodes returned in the response packet is logarithmic in the number of receivers whose request arrived in the same time interval. Assuming a 50 ms interval time (the sender would need to compute at most 20 signatures per second) and assuming that 1,000,000 receivers wanted to synchronize their time in that interval, the return packet would only need to contain 20 hash nodes or 200 bytes, assuming an 80 bit hash function. Any cryptographically secure hash function can be used for $H(x, y)$, for example MD5 [27], SHA-1 [17], or RIPEMD-160.

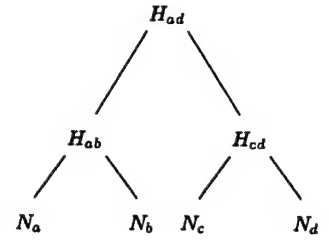


Figure 5: Hash tree over receiver nonces. Node $H_{ab} = H(N_a, N_b)$. $H_{ad} = H(H_{ab}, H_{cd})$.

4.2 DoS Attack on the Receiver

In this section, we discuss two DoS attacks on the client. Since we assume the attacker could have full control of the

network, some DoS attacks such as delay or drop packets are always possible. Delay packets could cause packets to violate the security condition and hence not to be authenticated. On the other hand, speeding up packets does not do anything at all. The receiver even benefits from this since she might be able to use a chain with a short disclosure delay that she could not use otherwise. We can show that replay packets cannot do much harm either. First, a duplicated packet is only accepted by the receiver within a short time period, since the security condition drops packets if they are replayed with a long delay. Second we can prevent the replay attack by adding a sequence number to each packet and by including the sequence number in the MAC. The TESLA protocol in the network layer or in the application layer will filter out duplicate packets.

In the rest of the subsection, we discuss some more complicated DoS attacks and show how to mitigate or prevent the attacks. First we discuss a flooding attack which fills up the receiver buffers. Second we discuss an attack that tries to waste the receiver's computation resources by unnecessarily re-computing the key chain.

DoS on the Packet Buffer

An powerful attack is to flood the multicast group with bogus traffic. This attack is serious because current multicast protocols do not enforce sending access control.[2] The solution we propose involves a weak but efficient and immediate authentication method that offers some protection against a flooding attack.

First if the receiver has a certain size buffer, we show that flooding cannot do much harm. Because the scheme only requires the receiver to buffer packets for the duration of one disclosure delay until the authenticity of the packets can be verified, hence the buffer size only needs to be the multiplication of the network bandwidth and the disclosure delay time. Assuming that the receiver has a 10Mbps network connection and a 500ms disclosure delay, the required buffer size is around 640kB, which should in general not be a major concern with today's workstations. Assuming 512byte network packets, the computation overhead to authenticate the packets is on the order of 1280 HMAC computations per second. Since the openssl HMAC-MD5 implementation processes on the order of 120,000 512-byte blocks per second on a 500MHz Pentium III Linux workstation, the estimated processor overhead for TESLA authentication is on the order of 1% of the CPU time.

Second if the receiver's buffer size is not large enough as computed above, flooding could result in a DoS attack because the receiver would drop packets due to a lack of buffer space.[3]

An obvious solution is to distribute a shared secret key to all receivers and to add a MAC to each packet with the shared secret key. This enables a receiver to quickly verify the packet, but it allows an attacker who knows the key to flood the clients anyhow.

Another approach is to use the key chain as a weak authentication method. Briscoe presents a related method for immediate authentication [8]. The receiver preauthenticates the packet by verifying that the disclosed key really is part of the key chain. Based on the disclosed key, the receiver can also immediately derive the time interval of the packet and also immediately verify the security condition. Both checks are efficient and do not require any additional space overhead in the packet. An attacker would need to receive a packet from the sender, extract the disclosed key, and use that key to flood the receivers. Fortunately, the flooding time period of each key is limited to one interval duration.

Yet another solution is to use the immediate authentication we propose in section 3.1. In this case, the message does not need to be added to a queue if it is immediately authenticated.

In practice, the receiver allocates a queue for each time interval to buffer incoming packets until they can be authenticated. If the receiver has too little memory to buffer all incoming traffic during the disclosure delay, it needs to decide on a drop or replacement policy in case of a full buffer. Dropping all packets of a particular interval once the buffer is full is a poor policy, because an attacker might insert the spoofed traffic only early in each time interval, causing the receivers to buffer mostly spoofed packets. Ideally, the receiver uses a random replacement policy once the buffer is full. For each incoming packet, the receiver picks a packet within the buffer to replace.

DoS on the Key Chain

Another DoS attack is specific to how the TESLA receiver reconstructs the key chain. If an attacker could fool a receiver to believe that a packet was sent out far in the future, and the receiver would try to verify the key disclosed in the packet by applying the pseudo-random function until the last committed key chain value. This attack can be easily prevented by checking that the packet interval is less or equal the latest interval that the sender can possibly be in. For an incoming packet sent in interval I_j, the receiver can verify if the interval I_j is not in the future, i.e. if the sender can already be in that interval. The ver-

[2] Source-Specific Multicast (SSM) is a new multicast protocol, and a new IETF working group was formed in August 2000 [22]. SSM tends to address this problem by enforcing that only one legitimate sender can send to the multicast group.

[3] We do not consider the flooding attack from a network perspective (where flooding can cause link congestion and results in dropping legitimate traffic) because any network protocol is susceptible to this attack.

ification condition is that $I_j < \lfloor (t_i - T_0)/T_{int} \rfloor$, where t_i is an upper bound on the sender's time that the receiver computes at the arrival of the packet.

5 Related Work

Researchers have proposed signing data packets to achieve source authentication. Since a digital signature achieves non-repudiation, a signature is much stronger than just authentication. As we mentioned in the introduction, the communication and computation overhead of current signature schemes is more expensive than schemes that are based on symmetric cryptography. We will review only the schemes that provide source authentication and not the schemes providing non-repudiation, i.e. [14, 29, 33, 25].

The earliest related work is by Cheung [11]. He proposes a scheme akin to the basic TESLA protocol to authenticate link-state routing updates between routers. He assumes that all the routers in a network are time synchronized up to $\pm\epsilon$, and does not consider the case of heterogeneous receivers.

Anderson et al. [1] present the Guy Fawkes protocol which provides message authentication between two parties. Their protocol has the drawback that it cannot tolerate packet loss. They propose two methods to guarantee that the keys are not revealed too soon. The first method is that the sender and receiver are in lockstep, i.e. the receiver acknowledges every packet before the sender can send the next packet. This severely limits the sending rate and does not scale to a large number of receivers. The second method to secure their scheme is to time-stamp each packet at a time-stamping service, which introduces additional complexity and overhead.

Canetti et al. propose to use k different keys to authenticate every message with k different MAC's for sender authentication [9]. Every receiver knows m keys and can hence verify m MAC's. The keys are distributed in such a way that no coalition of w receivers can forge a packet for a specific receiver. The communication overhead for this scheme is considerable, since every message carries k MAC's. The server must also compute k MACs before a packet is sent, which makes it more expensive than the scheme we present in this paper. Furthermore, the security of their scheme depends on the assumption that at most a bounded number (which is on the order of k) of receivers collude.

Briscoe proposes the FLAMeS protocol that is similar to the Cheung [11] and part of the basic TESLA protocol. Bergadano, Cavalino, and Crispo present an authentication protocol for multicast [5]. Their protocol is similar to Cheung [11] and to parts of the basic TESLA protocol.

Bergadano, Cavagnino, and Crispo, propose a protocol similar to the Guy Fawkes protocol to individually authenticate data streams sent within a group [4]. Their scheme requires that the sender receives an acknowledgment packet from each receiver before it can send the next packet. This prevents scalability to a large group. The advantage is that their protocol does not rely on time synchronization.

Unfortunately, their protocol is vulnerable to a man-in-the-middle attack. To illustrate the attack, we briefly review the protocol for one sender and one receiver (adapted to use the same notation as we established in this paper):

$$B \rightarrow A : KB_0, SN, \{KB_0, SN\}_{K_B^{-1}}$$
$$A \rightarrow B : A_1, MAC(KA_1, A_1), KA_0, SN, \{KA_0, SN\}_{K_A^{-1}}$$
$$B \rightarrow A : KB_1$$
$$A \rightarrow B : A_2, MAC(KA_2, A_2), KA_1$$

In their scheme, both A (the sender) and B (the receiver) pre-compute a key chain, KA_i and KB_i, respectively. In the following attack, B intends to authenticate data from A, but we will show that the attacker I can forge all data. The attacker I captures all messages from B and it can pretend to B that all the messages come from A. To A, the attacker I just pretends to be itself.

$$B \rightarrow I(A) : KB_0, SN, \{KB_0, SN\}_{K_B^{-1}}$$
$$I \rightarrow A : KI_0, SN, \{KI_0, SN\}_{K_I^{-1}}$$
$$A \rightarrow I : A_1, MAC(KA_1, A_1), KA_0, SN, \{KA_0, SN\}_{K_A^{-1}}$$
$$I \rightarrow A : KI_1$$
$$A \rightarrow I : A_2, MAC(KA_2, A_2), KA_1$$
$$I(A) \rightarrow B : A_1', MAC(KA_1, A_1'), KA_0, SN, \{KA_0, SN\}_{K_A^{-1}}$$

Note that the attacker I can forge the content of the message A_1 sent to B, because it knows the key KI_0. The attacker I can forge the entire subsequent message stream, without B noticing.

Another attack is that an eavesdropper that records a message exchange between A (sender) and B (receiver) can impersonate either A or B as a receiver to another sender C. This attack can be serious if the sender performs access control based on the initial signature packet and the revealed key chain. The attack is simple, the eavesdropper only needs to replay the initial signatures and all the disclosed keys collected.

6 Conclusions

In this paper, we have presented an extension to our TESLA scheme which provides a solution to the source authentication problem under the assumption that the

sender and receiver are loosely time synchronized. The basic TESLA protocol has the following salient properties:

- Low computation overhead. On the order of one MAC function computation per packet for both sender and receiver.
- Low communication overhead. Required is as little as one MAC value per packet. Periodically, the sender also needs to send out the secret keys.
- Perfect loss robustness. If a packet arrives in time, the receiver can verify its authenticity eventually (as long as it receives later packets).

The extensions we propose in this paper feature:

- The basic TESLA scheme provides delayed authentication. With additional information in a packet, we show in this paper how we can provide immediate authentication.
- We reduce the communication overhead when multiple TESLA instances with different authentication delays are used concurrently.
- We derive a tight lower bound on the disclosure delay.
- Harden the sender and the receiver against denial-of-service attacks.

7 Acknowledgments

We would like to thank Radia Perlman for the discussions on DoS attacks. We are also grateful to Bob Briscoe for helpful discussions and feedback.

References

[1] R. J. Anderson, F. Bergadano, B. Crispo, J.-H. Lee, C. Manifavas, and R. M. Needham. A new family of authentication protocols. *Operating Systems Review*, 32(4):9–20, October 1998.

[2] M. Bellare, R. Canetti, and H. Krawczyk. HMAC: Keyed-hashing for message authentication. Internet Request for Comment RFC 2104, Internet Engineering Task Force, Feb. 1997.

[3] S. Bellovin. The icmp traceback message. `http://www.research.att.com/~smb`, 2000.

[4] F. Bergadano, D. Cavagnino, and B. Crispo. Chained stream authentication. In *Selected Areas in Cryptography 2000*, Waterloo, Canada, August 2000. A talk describing this scheme was given at IBM Watson in August 1998.

[5] F. Bergadano, D. Cavalino, and B. Crispo. Individual single source authentication on the mbone. In *ICME 2000*, Aug 2000. A talk containing this work was given at IBM Watson, August 1998.

[6] N. Bhaskar and I. Kouvelas. Source-specific protocol independent multicast. Internet Draft, Internet Engineering Task Force, Mar. 2000. Work in progress.

[7] M. Bishop. A Security Analysis of the NTP Protocol Version 2. In *Sixth Annual Computer Security Applications Conference*, November 1990.

[8] B. Briscoe. FLAMeS: Fast, Loss-Tolerant Authentication of Multicast Streams. Technical report, BT Research, 2000. `http://www.labs.bt.com/people/briscorj/papers.html`.

[9] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Infocom '99*, 1999.

[10] R. Canetti, P. Rohatgi, and P.-C. Cheng. Multicast data security transformations: Requirements, considerations, and prominent choices. Internet draft, Internet Engineering Task Force, 2000. draft-data-transforms.txt.

[11] S. Cheung. An efficient message authentication scheme for link state routing. In *13th Annual Computer Security Applications Conference*, 1997.

[12] S. E. Deering. Host extensions for IP multicasting. Request for Comments (Standard) 1112, Internet Engineering Task Force, Aug. 1989.

[13] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol independent multicast-sparse mode (PIM-SM): protocol specification. Request for Comments (Experimental) 2362, Internet Engineering Task Force, June 1998.

[14] R. Gennaro and P. Rohatgi. How to Sign Digital Streams. Technical report, IBM T.J.Watson Research Center, 1997.

[15] N. Haller. The S/KEY one-time password system. Request for Comments (Informational) 1760, Internet Engineering Task Force, Feb. 1995.

[16] M. Handley, H. Holbrook, and I. Kouvelas. Protocol independent multicast - sparse mode (pim-sm): Protocol specification (revised). Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[17] U. S. Laboratory). Secure hash standard. Federal Information Processing Standards Publication FIPS PUB 180-1. `http://csrc.nist.gov/fips/fip180-1.txt` (ascii), `http://csrc.nist.gov/fips/fip180-1.ps` (postscript), Apr. 1995.

[18] L. Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11), Nov. 1981.

[19] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, J. Crowcroft, and B. Lueckenhoff. Asynchronous layered coding. a massively scalable reliable multicast protocol. Internet draft, Internet Engineering Task Force, July 2000. draft-ietf-rmt-pi-alc-01.txt.

[20] R. Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, 1980.

[21] D. L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. Internet Request for Comments, March 1992. RFC 1305.

[22] S.-S. Multicast. `http://www.ietf.org/html.charters/ssm-charter.html`.

[23] R. Perlman, C. Lee, T. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo, and M. Green. Simple multicast: A design for simple, low-overhead multicast. Internet

Draft, Internet Engineering Task Force, Mar. 1999. Work in progress.

[24] A. Perrig, R. Canetti, B. Briscoe, J. Tygar, and D. X. Song. TESLA: Multicast Source Authentication Transform. Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[25] A. Perrig, R. Canetti, J. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, May 2000.

[26] Reliable Multicast Transport (RMT). `http://www.ietf.org/html.charters/rmt-charter.html`.

[27] R. L. Rivest. The MD5 message-digest algorithm. Internet Request for Comments, Apr. 1992. RFC 1321.

[28] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[29] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *6th ACM Conference on Computer and Communications Security*, November 1999.

[30] Secure Multicast Group (SMuG). `http://www.ipmulticast.com/community/smug/`.

[31] D. X. Song and A. Perrig. Advanced and authenticatd marking schemes for ip traceback. Technical Report UCB/CSD-00-1107., UC Berkeley, July 2000.

[32] U. S. National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS), Federal Register 56. FIPS PUB 186, Aug. 1991.

[33] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. In *Proc. IEEE ICNP '98*, 1998.

Multicast Security: A Taxonomy and Some Efficient Constructions

Ran Canetti*, Juan Garay†, Gene Itkis‡, Daniele Micciancio§, Moni Naor¶, Benny Pinkas‖

Abstract—**Multicast communication is becoming the basis for a growing number of applications. It is therefore critical to provide sound security mechanisms for multicast communication. Yet, existing security protocols for multicast offer only partial solutions.**

We first present a taxonomy of multicast scenarios on the Internet and point out relevant security concerns. Next we address two major security problems of multicast communication: *source authentication*, and *key revocation*.

Maintaining authenticity in multicast protocols is a much more complex problem than for unicast; in particular, known solutions are prohibitively inefficient in many cases. We present a solution that is reasonable for a range of scenarios. Our approach can be regarded as a 'midpoint' between traditional Message Authentication Codes and digital signatures. We also present an improved solution to the key revocation problem.

I. Introduction

The popularity of multicast has grown considerably with the wide use of the Internet. Examples include Internet video transmissions, news feeds, stock quotes, software updates, live multiparty conferencing, on-line video games and shared whiteboards. Yet, security threats on the Internet have flourished as well. Thus the need for secure *and efficient* multicast protocols is acute.

Multicast security concerns are considerably more involved than those regarding point-to-point communication. Even dealing with the 'standard' issues of message authentication and secrecy becomes much more complex; in addition other concerns arise, such as access control, trust in group centers, trust in routers, dynamic group membership, and others.

A trivial solution for secure multicast is to set up a secure point-to-point connection between every two participants (say, using the IP-Sec protocol suite [17]). But this solution is prohibitively inefficient in most multicast scenarios. In particular, it obviates the use of multicast routing. Instead, we are looking for solutions that mesh well with current multicast routing protocols, and that have as small overhead as possible. In particular, a realistic solution must maintain the current way by which *data packets* are being routed; yet additional control messages can be introduced, for key exchange and access control.

This work. First, we present a taxonomy of multicast security concerns and scenarios, with a strong emphasis on IP multicast[1]. It soon becomes clear that the scenarios are so diverse that there is little hope for a unified security solution that accommodates all scenarios. Yet we suggest two 'benchmark' scenarios that, besides being important on their own, have the property that solutions for these scenarios may be a good basis in other settings. In a nutshell, one scenario involves a single sender (say, an on-line stock-quotes distributor) and a large number of recipients (say, hundreds of thousands). The second scenario is on-line virtual conferencing involving up to few hundreds of participants, where many (or all) of the participants may be sending data to the group.

Next we concentrate on a problem that emerges as a serious bottleneck in multicast security: *source* and *message* authentication. Known attempts to solve multicast security problems (e.g., [16], [22], [3], [28], [29], [21]) concentrate on the task of sharing a *single key* among the multicast group members. These solutions are adequate for encrypting messages so that only group members can decrypt. However, the single shared key approach is inadequate for source authentication, since a key shared among *all* members cannot be used to differentiate among senders in the group. In fact, the only known solutions for multicast authentication involve heavy use of public key signatures — and these involve considerable overhead, especially in the work needed to *generate* signatures.

We present solutions to the source authentication problem based on shared key mechanisms (namely, Message Authentication Codes — MACs), where each member has a *different* set of keys. We first present a basic scheme and then gradually improve it to a scheme that outperforms public-key signatures in several common scenarios. Our main savings are in the time to generate signatures.

The basic source authentication scheme for a single sender draws from ideas of [2], [11]: the sender holds a set of ℓ keys and attaches to each packet ℓ MACs – each MAC computed with a different key. Each recipient holds a subset of the ℓ keys and verifies the MAC according to the keys it holds. Appropriate choice of subsets insures that with high probability no coalition of up to w colluding bad members (where w is a parameter) know all the keys held by a good member, thus authenticity is maintained. We present several enhancements to this authentication scheme:

- A considerable gain in the computational overhead of the authentication scheme is achieved by noticing that the work needed for computing some known MAC functions on the same input and ℓ different keys is far less than the ℓ times the work to compute a single MAC. This is so since the message can first be hashed to a short string using key-less collision-resistant hashing.
- Using similar parameters to those of the basic scheme, one can guarantee that each good member has *many* keys that are

*IBM T.J. Watson research center. Email: canetti@watson.ibm.com

†Lucent Bell-Labs. Email: garay@research.bell-labs.com

‡News Data Systems. Email: gitkis@ndc.co.il

§Laboratory for Computer Science, MIT. Research supported by DARPA contract DABT63-96-C-0018. Email:miccianc@theory.lcs.mit.edu

¶Dept. of AM and CS, Weizmann Institute of Science. Research supported by ESPRIT working group RAND2. Email: naor@wisdom.weizmann.ac.il

‖Dept. of AM and CS, Weizmann Institute of Science. Research supported by an Eshkol Fellowship from the Israeli Ministry of Science. Email: bennyp@wisdom.weizmann.ac.il

[1] This survey is also the basis for a recent internet-draft [8], prepared for the Secure Multicast Group (SMuG) of the IRTF [26]. See also [15] for an earlier, more basic discussion of secure multicast issues.

known only to itself and to the sender. In order to break the scheme an adversary has to forge *all* the MACs computed with these keys. Thus it is enough that the sender attaches to the message only a *single bit* out of each generated MAC (as long as this bit cannot be successfully 'predicted' without knowing the key – see elaboration within). Consequently, the total length of the tag attached to the message can be reduced to only ℓ *bits*. (Also, such MAC functions may be more efficient than regular MACs.)

- A very similar method allows for *many senders* to use the same structure of keys — each sender will hold a different subset of keys, making sure that with high probability each sender-recipient pair shares a sufficient number of keys that are not known to any (small enough) bad coalition.
- It is further possible to increase security by making sure that no coalition of *senders* can forge messages, only large coalitions of recipients can. This property is beneficial when the recipients are relatively trusted (say, these are network routers). It is achieved by differentiating between *primary* and *secondary* keys. A sender only receives secondary keys, while primary keys are only held by the recipients. Each secondary key is derived by applying a pseudorandom function (e.g., a block cipher or keyed hash), keyed by the corresponding primary key, to the sender's public identity. Each recipient can now compute the relevant secondary keys and verify the MACs; yet, no coalition of senders knows even a single key other than its legitimate set of keys.

Finally, we consider the *membership revocation* problem. When a member leaves a multicast group it might be required to change the group key in a way that the leaving member does not learn the new key. A relatively efficient solution to this problem has been recently proposed [28], [29]. We present an improvement to this solution, that saves half of the communication overhead. (When a new member joins, the group might have to be re-keyed as well, in order to prevent the joining member from understanding previous group communication. This is a much simpler task: the group controller simply multicasts the new key encrypted with the previous group key.)

Organization. In Section II we list and discuss multicast security issues, in several common scenarios. In Section III we present our multicast authentication schemes, and in Section IV we present our improvements over past mechanisms for membership revocation.

II. Multicast Security Issues

We overview salient characteristics of multicast scenarios, and discuss the relevant security concerns. The various scenarios and concerns are quite diverse in character (sometimes they are even contradictory). Thus it seems unlikely that a single solution will be satisfactory for all multicast scenarios. This situation leads us to suggest two benchmark scenarios for developing secure multicast solutions.

Multicast group characteristics. We list salient parameters that characterize multicast groups. These parameters affect in a crucial way which security architecture should be used. The *group size* can vary from several tens of participants in small discussion groups, through thousands in virtual conferences and classes, and up to several millions in large broadcasts. *Member characteristics* include computing power (do all members have similar computing power or can some members be loaded more than others?) and attention (are members on-line at all times?).

A related parameter is *membership dynamics:* Is the group membership static and known in advance? Otherwise, do members only join, or do members also leave? How frequently does membership change and how fast should changes become effective? Also, is there a *membership control* center that has information about group membership? Finally what is the expected *life time* of the group (several minutes/days/unbounded)?

Next, what is the *number and type of senders*? Is there a single party that sends data? Several such parties? All parties? Is the identity of the senders known in advance? Are *non-members* expected to send data?

Another parameter is the *volume and type of traffic:* Is there heavy volume of communication? Must the communication arrive in real-time? What is the allowed latency? For instance, is it data communication (less stringent real-time requirements, low volume), audio (must be real-time, low volume) or video (real-time, high volume)? Also, is the traffic bursty?

Another parameter that may become relevant is the routing algorithm used. For instance, a security mechanism may interact differently with dense-mode and sparse-mode routing. Also, is all routing done via a single server or is it distributed?

Security requirements. The most basic security requirements are secrecy and authenticity. *Secrecy* usually means that only the multicast group members (and all of them) should be able to decipher transmitted data. We distinguish two types of secrecy: *Ephemeral secrecy* means preventing non group-members from easy access to the transmitted data. Here a mechanism that only delays access may be sufficient. *Long-term secrecy* means protecting the confidentiality of the data for a long period of time. This type of secrecy is often not needed for multicast traffic.

Authenticity may take two flavors: *Group authenticity* means that each group member can recognize whether a message was sent by a group member. *Source authenticity* means that it is possible to identify the particular sender within the group. It may be desirable to be able to verify the origin of messages even if the originator is not a group member.

Other concerns include several flavors of *anonymity* (e.g., keeping the identity of group members secret from outsiders or from other group members, or keeping the identity of the sender of a message secret). A related concern is protection from *traffic analysis*. A somewhat contradictory requirement is *non-repudiation,* or the ability of receivers of data to prove to third parties that the data has been transmitted.

Access control, or making sure that only registered and legitimate parties have access to the communication addressed to the group, is usually obtained by maintaining ephemeral secrecy of the data. Enforcing access control also involves authenticating potential group members. The access control problem becomes considerably more complex if members may join and leave with time.

Lastly, maintaining *service availability* is ever more relevant in a multicast setting, since clogging attacks are easier to mount and are much more harmful. Here protection must include multicast-enabled routers as well as end-hosts.

Trust issues. In simple scenarios there is a natural *group owner* that can be trusted to manage the group security. Typical roles are access control, logging traffic and usage, and key management. (It may be convenient, but not necessary, to identify the group owner with the *core* used in some multicast routing protocols, e.g. in [3].) In other cases no single entity is totally trusted; yet different entities can be trusted to perform different tasks (for instance, the access-control entity may be different than the entity that distributes keys). In addition, basing the security of the entire group on a single service makes the system more vulnerable. Thus it is in general beneficial to distribute the security tasks as much as possible.

A natural approach for distributing trust in multicast security centers is to use *threshold cryptography* [9], [13] and *proactive security* [7] techniques to replace a single center with a distributed service with no single point of failure. This is an interesting topic for future research.

Performance. Performance is a major concern for multicast security applications. The most immediate costs that should be minimized are the *latency* and *work overhead* per sending and receiving data packets, and the *bandwidth overhead* incurred by inflating the data packets via cryptographic transformations. *Secure memory* requirement (e.g., lengths of keys) is a somewhat less important resource, but should also be minimized. Here distinction should be made between the load on strong server machines and on weak end-users.

Other performance overheads to be minimized include the *group management* activity such as group initialization and member addition and deletion. Here member deletion may cause severe overhead since keys must be changed in order to ensure revocation of the cryptographic abilities of the deleted members. We elaborate in Section IV.

An additional concern is possible congestion, especially around centralized control services at peak sign-on and sign-off times. (A quintessential scenario is a real-time broadcast where many people join right before the broadcast begin and leave right after it ends.) Another performance concern is the work incurred when a group member becomes active after being dormant (say, off-line) for a while.

Benchmark Scenarios

As seen above, it takes many parameters to characterize a multicast security scenario, and a large number of potential scenarios exist. Each scenario calls for a different solution; in fact, the scenarios are so different that it seems unlikely that a single solution will accommodate all. This is in sharp contrast with the case of unicast security, where a single architectural approach (public-key based exchange of a key, followed by authenticating and encrypting each packet using derived keys) is sufficient for most scenarios.

In this section we present two very different scenarios for secure multicast, and sketch possible solutions and challenges. These scenarios seem to be the ones that require most urgent solutions; in addition, they span a large fraction of the concerns described above, and solutions here may well be useful in other scenarios as well. Thus we suggest these scenarios as benchmarks for evaluating security solutions.

Single source broadcast. Consider a single source that wishes to continuously broadcast data to a large number of recipients (e.g. a news agency that broadcasts news-feeds and stock-quotes to paying customers). Such applications are common in the Internet today, but they still typically rely on unicast routing and have few or no security protections.

Here the number of recipients can be hundreds of thousands or even millions. The source is typically a top-end machine with ample resources. It can also be parallelized or even split into several sources in different locations. The recipients are typically lower-end machines with limited resources. Consequently, any security solution should be optimized for efficiency at the recipient side.

Although the life-time of the group is usually very long group membership is typically dynamic: members join and leave at a relatively high rate. In addition, at peak times (say, before and after important broadcasts) a high volume of sign-on/sign-off requests are expected.

The volume of transmitted data may change considerably: if only text is being transmitted then the volume is relatively low (and the latency requirements are quite relaxed); if audio/video is transmitted (say, in on-line pay-TV) then the volume can be very high and very little latency is allowed.

Authenticity of the transmitted data is a crucial concern and should be strictly maintained: a client must never accept a forged stock-quote as authentic. Another important concern is preventing non-members from using the service. This can be achieved by encrypting the data; yet the encryption may be weak since there is no real secrecy requirement, only prevention from easy unauthorized use. Regarding trust, here there is typically a natural group owner that manages access-control as well as key management. However, the sender of data may be a different entity (say, Yahoo! broadcasting Reuters news).

A natural solution for this scenario may have a group management center that handles access control and key management. (To scale the solution to a larger number of recipients the center can be distributed, or a hierarchal structure can be introduced.) It is stressed that the center handles only 'control traffic'. The data packets are routed using current multicast routing protocols. Encryption can be done using a single key shared by all members. Yet, two main cryptographic problems remain: How to authenticate messages, and how to make sure that a leaving member loses its ability to decrypt.

A simple and popular variant of this scenario, file transmission and updates, typically has static group membership and does not require on-line delivery of data.

Virtual Conferences. Typical virtual conference scenarios include on-line meetings of corporate executives or committees, town-hall type meetings, interactive lectures and classes, and multiparty video games. A virtual conference involves several tens to hundreds of peers, often with roughly similar computational resources. Usually most, or all, group members may a-priori wish to transmit data (although often there is a small set of members that generate most of the bandwidth).

The group is often formed per event and is relatively short-lived. Membership is usually static: members usually join at start-up, and remain signed on throughout. Furthermore, even if a member leaves, cryptographically disconnecting it from the

group is often not crucial. Bandwidth and latency requirements vary from application to application, similarly to the case of single source broadcast.

Authenticity of data *and sender* is the most crucial security concern. In some scenarios maintaining secrecy of data and anonymity of members may be crucial as well; in many other scenarios secrecy of data is not a concern at all. Although there is often a natural group owner that may serve as a trusted center, it beneficial to distribute trust as much as possible.

Also here a simple approach to a solution uses a server that handles access control and key management. Encryption, when needed, can be dealt with as above. Yet, the performance requirements from the authentication mechanism are very different. In particular, in contrast with the single sender scenario, here signing data packets may be prohibitively slow on the sender's machine. In addition, there are far less receivers, and the group members may be somewhat more trustworthy. Virtual conferencing applications are also typically more tolerant to occasional and local authentication errors. These considerations point to an alternative approach to solving the multicast authentication problem. In the next section we describe this alternative approach.

III. Efficient Authentication Schemes

We concentrate on two approaches to authentication: public key signatures, and MACs. (We do not address information-theoretic authentication mechanisms, such as [10], [25], [6], which are inherently inefficient for groups of non-trivial size.)

Public key signatures are perhaps the most natural mechanism for multicast authentication. Yet, signatures are typically long, and computing and verifying each signature requires a significant computational overhead. Applying signatures to authenticate streams of data was investigated in [14], who proposed a chaining mechanism that requires a single signature per stream. These constructions do not tolerate packet loss, and are thus incompatible with IP multicast. Alternatively, [30] suggested using tree-based hashing to authenticate streams. This approach is a little less efficient, and incurs some latency, but it better tolerates packet loss.

As an alternative to public key signatures, we propose an authentication method based on *message authentication codes* (MACs). A MAC is a function which takes a secret key k and a message M and returns a value $\mathrm{MAC}(k, M)$. Very informally, a MAC scheme is *unforgeable* if an adversary that sees a sequence $\{M_i, \mathrm{MAC}(k, M_i)\}$ where the M_i's are adaptively chosen, but does not know k, has a negligible probability to generate $\mathrm{MAC}(k, M)$ for *any* $m \notin \{M_i\}$.

While MACs are typically much more efficient to generate and verify than digital signatures, they require that all potential verifiers have access to a shared key, k. This property makes MACs seemingly insufficient for achieving source authentication: any potential receiver who has the key k can "impersonate" the sender. We present new MAC-based authentication methods which achieve source authentication, and are more efficient than public key based authentication (especially in the time to generate signatures). We first present a description of a basic scheme, followed by several variants and improvements (see sketch in the Introduction).

We analyze the following salient resources for all the schemes we present: The *running time* required to authenticate a message and to verify an authentication, denoted T_S and T_V, respectively. The *length of the keys* that the authenticator and the verifier should store, denoted M_S and M_V, respectively. The *length of the authentication message* (the MAC or the signature), denoted C. These resources are obviously related to the latency, secure memory and bandwidth overhead parameters discussed in Section II.

Per-message unforgability of MAC schemes. We distinguish between two types of attacks against a MAC scheme. One is a complete break, where the attacker can authenticate *any* message of its choice (e.g., a key recovery attack). The other attack allows the attacker to *randomly* authenticate false messages; here the attacker can authenticate a given message with some fixed and small probability (but does not know a-priori whether it will be able to authenticate the message). Our schemes do not allow complete break with higher probability than the underlying MAC scheme. Yet, we do allow for random authentication errors with non-negligible probability (say, 2^{-20} up to 2^{-10}).

A bit more formally, we say that a MAC scheme is *q-per-message unforgeable* if no (probabilistic polynomial-time) adversary has a positive expected payoff in the following guessing game: the adversary can ask to receive the output of the MAC on a sequence of messages $m_1, \ldots, m_k$ of its choice, and then decide to quit or to gamble. If it quits it receives a payment of \$0. Otherwise, it chooses a message $m \notin \{m_1, \ldots, m_k\}$ and tries to guess the value of the MAC on m. The adversary receives $\$(1-q)$ if its guess is correct, and pays $\$q$ otherwise. In other words the adversary may guess correctly the value of the MAC with probability at most q, but (except with negligible probability) won't "know" whether its guess is correct.

We believe that for most systems a small (although non-negligible) per-message unforgeability (say, $q = 2^{-20}$) is sufficient. Note that per-message unforgeability is a weaker security property than standard unforgeability, in the sense that any scheme that is unforgeable in the standard sense is also q-per-message unforgeable (for any non-negligible value of q). The converse does not necessarily hold.

A. The Basic Authentication Scheme for a Single Source

Let w be the maximum number of corrupted users. The basic scheme proceeds as follows:

- The source of the transmissions (S) knows a set of $\ell = e(w+1)\ln(1/q)$ keys, $R = \langle K_1, \ldots, K_\ell \rangle$.
- Each recipient u knows a subset of keys $R_u \subset R$. Every key K_i is included in R_u with probability $1/(w+1)$, independently for every i and u [2].
- Message M is authenticated by S with each key K_i using a MAC and $\langle \mathrm{MAC}(K_1, M), \mathrm{MAC}(K_2, M), \ldots, \mathrm{MAC}(K_\ell, M) \rangle$ is transmitted together with the message.
- Each recipient u verifies all the MACs which were created using the keys in its subset R_u. If any of these MACs is incorrect then u rejects the message.

[2]Notice that this can be accomplished by using a $(w+1)$-wise independent mapping from users to subsets.

The performance parameters are the following. The source must hold $M_S = \ell = e(w+1)\ln(1/q)$ basic MAC keys. Each receiver expects to hold $M_V = e\ln(1/q)$ MAC keys.[3] The communication overhead per message is $C = e(w+1)\ln(1/q)$ MAC outputs. The running time overhead is $T_S = e(w+1)\ln(1/q)$ MAC computations for the source and only $T_V = e\ln(1/q)$ MAC computations for a receiver.

Theorem 1: Assume that the probability of computing the output of a MAC without knowing the key is at most q'. Let u be a user. Then the probability that a coalition of w corrupt users can authenticate a message M to u is at most $q + q'$ (the probability is taken over the choice of key subsets and over the message)[4].

Proof (sketch): For every user u and any coalition of w users, the probability that a specific key is good (i.e. contained in a user's subset, but not in the subset of any of the w members of the coalition) is ' $g = \frac{1}{w+1}\left(1-\frac{1}{w+1}\right)^w = \frac{1}{(w+1)\left(1+\frac{1}{w}\right)^w} > \frac{1}{e(w+1)}$. Therefore, the probability that R_u is completely covered by the subsets held by the coalition members is $(1-g)^\ell < (1-\frac{1}{e(w+1)})^{e(w+1)\ln(1/q)} < e^{-\ln(1/q)} = q$. If R_u is not covered, the set $\{MAC(K_i, M)\}_{i\in R_u}$ contains at least one MAC for which the coalition does not know K_i. The probability of computing it correctly is at most q'. By union bound, the probability that the coalition can authenticate M to u is at most $q + q'$. □

Notice that when the keys of a user are not covered by the coalition, the coalition cannot check in advance (off-line) whether it can authenticate a specific message. Therefore the probability q' of authenticating a message by breaking a MAC can be rather large (e.g. even $q' = 2^{-10}$ might be reasonable for many applications).

A nice feature of this construction is that the complexity does not depend on the total number of parties but rather only on the maximum size of a corrupt coalition and the allowed error probability. We remark that a similar idea was previously used by Fiat and Naor for broadcast encryption (described in [2]) and by Dyer et al. for pairwise encryption [11].

The security is against an arbitrary, but fixed, coalition of up to w corrupt recipients. Notice that it is possible to construct schemes which are secure against any coalition of size w as follows. Let $q = (n\cdot\binom{n}{w})^{-1}$ (i.e. 1 over the number of possible combinations of coalitions and users). By a probabilistic argument, there exists a system for n recipients in which the subset of no user is covered by the union of the subsets of a coalition of size w. The system has a total of less than $e(w+1)^2\ln n)$ keys, and each recipient has a subset of expected size less than $e(w+1)\ln n$.

B. Smaller Communication Overhead

We now describe a scheme with a lower communication overhead. The idea behind it is that using just four times as many keys as in the basic scheme, one can ensure that the coalition does not know $\log(1/q)$ of the user's keys. Each key can therefore be used to produce a MAC with a *single bit output* and the communication overhead is improved. The coalition would have to guess $\log(1/q)$ bits to create a false authentication and its probability of success is as before.

Recall the basic scheme: it limits the success probability of a corrupt coalition to be $q + q'$, where q' is the per-message unforgeability. The MAC output must be at least $\log_2(1/q')$ bits long. Therefore, assuming $q' = q$, the communication overhead is $C > e(w+1)\ln^2(1/q)$ *bits*. The improved scheme achieves a communication overhead smaller than $4e(w+1)\ln(1/q)$ *bits*.

The improved scheme uses a MAC with a single bit output. (Current constructions of MACs have much larger outputs, but our schemes can use a single bit of this output. It might also be possible to design a special-purpose MAC function with a single bit output, which would be more efficient than standard constructions.) For simplicity of exposition, assume that for this MAC $q' = 1/2$. If the keys of a corrupt coalition do not cover $\log(1/q)$ keys of a user's subset, then the probability that the user accepts an unauthentic message from the coalition is at most[5] q. In the suggested scheme the source uses $\ell = 4e(w+1)\ln(1/q)$ keys where each key is included in a user's subset with probability $1/(w+1)$.

All performance parameters are multiplied by four. The source must store $M_S = \ell = 4e(w+1)\ln(1/q)$ basic MAC keys. Each receiver expects to store $M_V = 4e\ln(1/q)$ basic MAC keys. The communication overhead is $C = 4e(w+1)\ln(1/q)$ *bits* per message. The source must compute $4e(w+1)\ln(1/q)$ MACs, whereas each receiver expect to compute $4e\ln(1/q)$ MACs.

Theorem 2: Consider a MAC with a single bit output that is $\frac{1}{2}$-per-message unforgeable, and consider the above scheme using this MAC and $\ell = 4e(w+1)\ln(1/q)$ keys. Then for every user u and coalition of other w corrupt users, it holds with probability $1-q$ that the resulting scheme is q-per-message unforgeable (with respect to the coalition and u).

Proof (sketch): The probability that a specific key is good is $g > \frac{1}{e(w+1)}$ as before. Since the MAC is $\frac{1}{2}$-per-message unforgeable, the coalition cannot guess with probability better than $1/2$ the output of a MAC whose key it does not know. Therefore the expected success probability of a corrupt coalition is $\sum_{i=0}^{\ell}\binom{\ell}{i}g^i(1-g)^{\ell-i}2^{-i} = (1-g/2)^\ell < q^2$. By Markov inequality, with probability at most q the coalition has a probability greater than q to compute all MACs with key in R_u. In other words, with probability $1-q$ the scheme is q-per-message unforgeable. □

C. Multiple Dynamic Sources

The schemes presented above can be easily extended to enable *any* party to send authenticated messages. The global set of ℓ keys is $w+1$ times bigger than in the single source scheme, and every party receives a random subset R_u of these keys. Keys

[3] A straightforward modification of this scheme allows each member to have a *fixed* number of keys.

[4] A similar result holds with respect to per-message unforgability. That is, if the MAC is q'-per-message unforgeable then for any user u and coalition of other w corrupt users, it holds with probability $1-q$ that the resulting scheme is q'-per-message unforgeable with respect to the coalition and the user.

[5] More formally, assume that it is not possible to distinguish in polynomial time between the output of the MAC and a random bit with probability better than $1/2+\varepsilon$. Then (see [23]) one can use a "hybrid argument" to show that it is not possible to distinguish between m MAC outputs and an m bit random string with probability better than $1/2 + m\varepsilon$.

are included R_u independently at random with probability $\frac{1}{w+1}$. When a party u sends a message, it authenticates it with all the keys in R_u, and every receiving party v verifies the authentications that were performed with the keys in $R_u \cap R_v$. It is straightforward to verify that the resulting schemes are as secure as the single source schemes. Note that the (average) communication and computation overheads are not changed. The mapping of users to subsets can be done with a public $(w+2)$-wise independent hash function.

Following, we present a better method which supports a *dynamic* set of sources and has the following properties:

- The total number of keys is as in schemes for a single source, but every party can send authenticated messages.
- The scheme does not require the set of sources to be defined in advance or to contain all parties. Rather, it allows to *dynamically* add sources.
- The scheme distinguishes between the set of sources and the set of receivers. Only coalitions of more than w *receivers* can send false authenticated messages. The keys of sources do not help such coalitions. This property is especially useful if receivers are more trusted than senders, as might be the case for example if the receivers are network routers.
- The scheme provides a computational (rather than an information theoretic) security against revealing to a coalition all the keys in the intersection of a source and a receiver's subsets.

The scheme uses a family of pseudo-random functions $\{f_k\}$ (see [20] for a discussion of pseudo-random functions). It is based on a single source scheme and can be built upon the basic scheme we described in Section III-A or the communication efficient scheme of Section III-B.

Initialization: The scheme uses ℓ *primary keys* $\langle k_1, \ldots, k_\ell \rangle$, where ℓ is as in the single source schemes ($\ell = O(w \log(1/q))$. Each key k_i defines a pseudo-random function f_{k_i}.

Receiver Initialization: Each party v which intends to receive messages obtains a subset R_v of primary keys. Every primary key k_i is included in R_v with probability $1/(w+1)$.

Source Initialization: Every party u which wishes to send messages receives a set of *secondary keys* $S_u = \langle f_{k_i}(u), f_{k_2}(u), \ldots, f_{k_\ell}(u) \rangle$. This set can be sent any time after the system has been set-up, and the identity or the number of sources does not have to be defined in advance.

Message Authentication: When a party u sends a message M it authenticates it with all the secondary keys in S_u. That is, $\forall\, k \in S_u$ it computes and attaches a MAC of M with k.

Every receiving party v computes all the secondary keys of u with primary key in R_v. Namely, it computes the set $f_{R_v}(u) = \{f_k(u) | k \in R_v\}$. It then verifies all the MACs which were computed using these keys.

The number of keys which are used and stored is as in the single source scheme. The work of the sources is as in the previous schemes, and receivers only have the additional task of evaluating f to compute a secondary key for each of the primary keys in their subset

A very useful property of this scheme is that it enables a *dynamic* set of sources. New parties can be allowed to send authenticated messages by giving them a corresponding set of secondary keys. Another useful property of the scheme is that the set of sources can be separated from the set of receivers, and no coalition of sources can break the security. It also enables to give sources dedicated keys for authenticating different messages. An attractive application of these properties is to give the source which is designated to broadcast at time T the set of secondary keys $f_k(T)$, and require it to use them to authenticate its broadcast at that time. This approach ensures that sources can only send information to the group in their designated time slots.

D. Signatures vs. MACs: a rough performance comparison

Compared to the performance of public key signatures, our authentication schemes dramatically reduce the running time of the authenticator. The running time of the verifier and the communication overhead are of the same order as public key signatures (the exact comparison depends on the size of the corrupt coalitions against which the schemes operate).

Consider for example RSA signatures with an 1024 bit modulus. Recent measurements indicate that on a fast machine (200MHz power pc) a signature (authentication) takes $T_S = 1/50$s and verification time is $T_V = 1/30,000$s.[6] For 768-bit DSS on a similar platform the numbers are roughly $T_S = 1/40$ and $T_V = 1/70$. In comparison, an application of the compression function of MD5 takes about $1/500,000$ of a second; an application of DES takes roughly the same time. Future block ciphers and hash functions are expected to be considerably faster.

The schemes we introduce require the parties to apply many MACs with different keys to the same message. Current constructions of MACs achieve both a hash down of the input to the required output size, and a keyed unpredictable output. For the suggested schemes it is preferable to perform a *single* hash down of the message, and then compute MACs of the hashed down value[7]. Regarding HMAC [19], [4] as a reference MAC function, this implies that only one of HMAC's two nested keyed applications of a hash function should be used (in the terms of [4] this corresponds to defining ℓ MACs with keys $k_1, \ldots, k_\ell$ as $\{NMAC_{k,k_i}\}_{i=1}^{\ell}$, where the key k is common to all functions). Therefore in comparisons to public key operations we assume that a MAC takes a single application of a compression function of the hash function in use (say, MD5), or equivalently a single application of a block-cipher such as DES.

Furthermore, we believe that more efficient MACs could be designed for our authentication schemes. In particular, these MAC functions would make use of the fact that they can have a single bit output, and would have small amortized complexity (for evaluations of the function on the same input and many keys). Authentication schemes based on such functions should be considerably more efficient than schemes based on HMAC.

Table I compares the overhead of RSA and DSS signatures to the overhead of the suggested authentication schemes with some specific parameters. The communication overhead of the basic and improved schemes are based on using only 10 bits out of each MAC.

The table describes the number of authentications and verifications that can be performed per second, the communication

[6] The numbers here are for highly optimized RSA code with verification exponent 3. Verification using standard RSA code is considerably slower.

[7] The initial hash down is also performed for public key signatures, since messages should be reduced to the size of the public key modulus. Therefore we omit its computation time from the running time overhead of our schemes.

	Auth.	Ver.	Comm.	Source Key	Receiver Key
Units	(ops/sec)	(ops/sec)	(bits)		
RSA[6] 1024 bits	50	30,000	1024	2048 bits	1024 bits
DSS, 768 bits	70	40	1536	1536 bits	1536 bits
Basic scheme, $w = 10, q = 10^{-3}$	2,650	26,500	1900	190 MAC keys	19 MAC keys
Low Comm., $w = 10, q = 10^{-3}$	660	6,600	760	760 MAC keys	76 MAC keys
Perfect Sec., $n = 10^4, q' = 10^{-3}$	200	2000	25,000	2500 MAC keys	250 MAC keys

TABLE I

A PERFORMANCE COMPARISON OF AUTHENTICATION SCHEMES.

overhead in bits, and the length of the key used by the source and the receivers. The first two rows are for RSA and DSS signatures. The third row provides an estimate for our basic authentication scheme, providing per-message unforgeability of $q = 10^{-3}$ against coalitions of up to ten corrupt users. Next we present the performance of the communication efficient variant, in which each MAC has a single bit output. Last is the performance of a scheme which guarantees that no coalition knows all the keys of any user (its overhead seems too large to justify its use).

It is seen that the signing time is much shorter in our scheme than with public key signatures. The verification time is comparable to (highly optimized) RSA and much faster than DSS.[8]

IV. Dynamic Secrecy – User Revocation

Secret group communication can be achieved by encrypting messages with a group key. This raises the question of how to add or remove users from the group. When a new member joins the group, the common key can be sent to the new member using secure unicast. Alternatively, if the previous communications should be kept secret from the new user, a new common key can be generated and sent to the old group members (encrypted with the old common key) and to the new member (using secure unicast). User deletion is more problematic. Obviously, it is not enough to just ask members who leave the group to delete their group key, and it is essential to change the key with which group communication is encrypted in order to conceal future communications from former group members. This problem is known as *user revocation* or *blacklisting*, and is particularly important in applications like pay-per-view in which only paying customers should be allowed to receive transmissions.

We survey some solutions for the member deletion problem, describe a particularly appealing construction from [28], [29] based on binary trees, and present an improved construction with reduced communication overhead. We also show how our construction is more resistant to a certain kind of attack.

A. Some User Revocation Schemes

A trivial solution for the member revocation problem is for each group member to share a individual secret key with a center which controls the group. When a member is deleted from the group, the center chooses a new common key to encrypt future multicast messages, and sends it to every group member, encrypted with the respective individual secret keys. This solution does not scale up well since a group of n members requires a key renewal message with $n - 1$ new keys.

A more advanced solution was suggested by Mittra [22]. It divides the multicast group into subgroups which are arranged in a hierarchical structure and each has a special group controller. The user revocation overhead is linear in the size of a subgroup. However, this solution introduces group controllers in every subgroup which form many possible points of failure, both for availability and for security.

There are also suggestions to use public key technology, namely generalized Diffie-Hellman constructions, to enable communication efficient group re-keying (e.g. [27]). However, for a group of n members these suggestions require $O(n)$ exponentiations. For most applications this overhead is far too high to be acceptable in the near future.

A totally different solution was suggested by Fiat and Naor [12] and was motivated by pay-TV applications. It enables a single source to transmit to a dynamically changing subset of legitimate receivers from a larger group of users, such that coalitions of at most k users cannot decrypt the transmissions unless one of them is a member in the subset of legitimate receivers. A very nice feature of this scheme is that the overhead of a rekeying message does not depend on the number of users that are removed from the group. The communication overhead of the scheme is $O(k \log^2 k \log(1/p))$, where p is an upper bound on the probability that a coalition of at most k users can decrypt a transmission to which it is not entitled. The scheme also requires each user to store $O(\log k \log(1/p))$ keys. The main drawback in applying it for Internet applications is that the security is only against coalitions of up to k users, and the parameter k substantially affects the overhead of the scheme. It should also be noted that this scheme is only suitable for a single source of transmission, but this obstacle might be overcome if all users trust the owner of the group and all communication is sent through a unicast channel to this owner and from there multicasted to the group (as is the case for example in CBT routing).

B. A Tree Based Scheme

Tree based group rekeying schemes were suggested by Wallner et al. [28] (who used binary trees), and independently by Wong et al. [29] (who consider the degree of the nodes of the tree as a parameter). We concentrate on the scheme of [28] since

[8] In addition note that if public key signatures are used for authentication then each receiver should store the verification keys of all sources, or alternatively the verification keys should be certified by a certification authority and then the length of the authentication message and the verification times are doubled.

it requires a smaller communication overhead per user revocation. This scheme applied to a group of n users requires each user to store $\log n+1$ keys. It uses a message with $2\log n-1$ key encryptions in order to delete a user and generate a new group key. This process should be repeated for every deleted user. The scheme has better performance than the Fiat-Naor scheme when the number of deletions is not too big. It is also secure against any number of corrupt users (they can all be deleted from the group, no matter how many they are). A drawback of this scheme is that if a user misses some control packet relative to a user deletion operation (e.g., if it temporarily gets disconnected from the network), it needs to either ask for all the missed control packets, or incur in a communication overhead comparable to a user addition operation.

We now describe the scheme of [28]. Let $u_0, \ldots, u_{n-1}$ be n members of a multicast group (in order to simplify the exposition we assume that n is a power of 2). They all share a group key k with which group communication is encrypted. There is a single group controller, which might wish at some stage to delete a user from the group and enable the other members to communicate using a new key k', unknown to the deleted user.

The group is initialized as follows. Users are associated to the leaves of a tree of height $\log n$ (see Figure 1). The group controller associates a key k_v to every node of the tree, and sends to each user (through a secure channel) the keys associated to the nodes along the path connecting the user to the root. For example, in the tree of Figure 1, user u_0 receives keys k_{000}, k_{00}, k_0 and k. Notice that the root key k is known to all users and can be used to encrypt group communications.

In order to remove a user u from the group, the group controller performs the following operations. For all nodes v along the path from u to the root, a new key k'_v is generated. New keys are encrypted as follows. Key $k'_{p(u)}$ is encrypted with key $k_{s(u)}$, where $p(u)$ and $s(u)$ denote respectively the parent and sibling of u. For any other node v along the path from u to the root (excluded), key $k'_{p(v)}$ is encrypted with keys k'_v and $k_{s(v)}$. All encryptions are sent to the users. For example, in order to remove user u_0 from the tree of Figure 1 the following encryptions are transmitted (see Figure 2): $E_{k_{001}}(k'_{00}), E_{k'_{00}}(k'_0), E_{k_{01}}(k'_0), E_{k'_0}(k'), E_{k_1}(k')$. It is easy to verify that each user can decrypt only the keys it is entitled to receive.

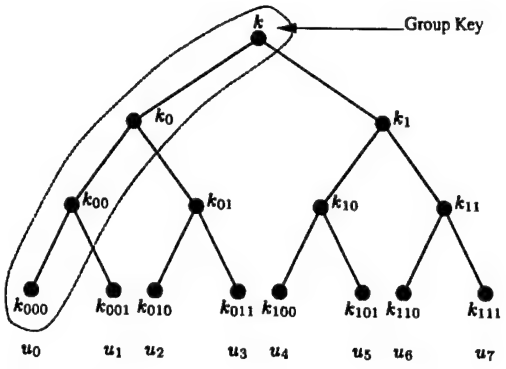


Fig. 1. The tree key data structure (the keys of u_0 are encircled).

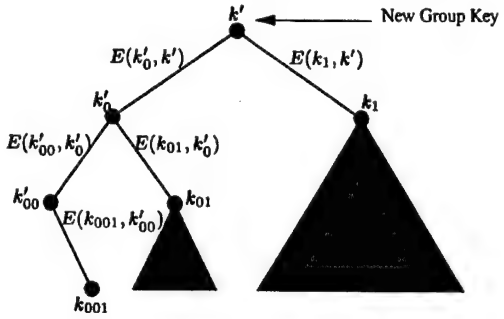


Fig. 2. Key revocation in the basic scheme.

C. The Improved Scheme

The improved scheme reduces the communication overhead of [28] by a factor of two, from $2\log n$ to only $\log n$. The initialization of the scheme is the same as in [28]. We now describe the user revocation procedure. Let G be pseudo-random generator which doubles the size of its input [5]. Denote by $L(x), R(x)$ the left and right halves of the output of $G(x)$, i.e., $G(x) = L(x)R(x)$ where $|L(x)| = |R(x)| = |x|$. To remove a user u, the group controller associates a value r_v to every node v along the path from u to the root as follows: It chooses $r_{p(u)} = r$ at random and sets $r_{p(v)} = R(r_v) = R^{|u|-|v|}(r)$ for all other v (where $p(v)$ denotes the parent of v). The new keys are defined by $k'_v = L(r_v) = L(R^{|u|-|v|-1}(r))$. Notice that from r_v, one can easily compute all keys $k'_v, k'_{p(v)}, k'_{p(p(v))}$ up to the root key k'. Finally each value $r_{p(v)}$ is encrypted with key $k_{s(v)}$ (where $s(v)$ denotes the sibling of v) and sent to all users. For example, in order to remove user u_0 from the tree of Figure 1, we send encryptions $E_{k_{001}}(r), E_{k_{01}}(R(r)), E_{k_1}(R(R(r)))$. One can easily verify that, under the assumption that G is a cryptographically strong pseudo-random generator, each user can compute from the encryptions all and only the keys it is entitled to receive.

Advantages of the new scheme: This construction halves the communication overhead of the basic scheme to only $\log n$, and its security can be rigorously proven. It has an additional advantage: In the scheme of Wallner et al the group controller chooses the group key (the root key), whereas is our construction this key is the output of a pseudo-random generator. Sup-

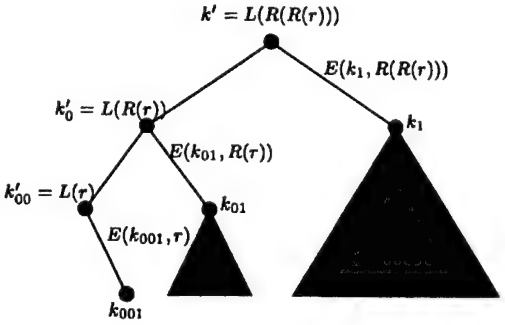


Fig. 3. Key revocation in the improved scheme.

pose that there is an adversary which can break encryptions performed with a subset of the key space (for example keys in which certain bits have a linear dependency), and furthermore that this adversary has gained temporary control over the group controller (e.g. when the controller was manufactured). Then if the scheme of [28] is used, the adversary might corrupt the method by which the group controller generates keys in such a way that the root key would always be chosen from the "weak" subspace. However, if our scheme is used, and the pseudo-random generator $G(x) = L(x)R(x)$ is cryptographically strong, then it will be hard to find values r such that the root key $k = L(R(R(\cdots(r)\cdots)))$ is weak.

Independently, McGrew and Sherman [21] have presented a tree based rekeying scheme which has the same overhead as ours. However, the security of their scheme is based on non-standard cryptographic assumptions and is not rigorously proven. In comparison, the security of our scheme can be rigorously proven based on the widely used assumption of the existence of pseudo-random generators [5].

Acknowledgments

We are very thankful to Shai Halevi, Rosario Gennaro and Tal Rabin, for discussions on the problems and possible solutions for multicast security.

REFERENCES

[1] A. Albanese, J. Bloemer, J. Edmonds, M. Luby and M. Sudan, "Priority Encoding Transmission", *IEEE Tran. on Inf. Th.*, Vol. 42, No. 6, Nov. 1996.

[2] N. Alon, "Probabilistic Methods in Extremal Finite Set Theory", in *Extremal Problems for Finite Sets*, 1991, 39–57.

[3] A. J. Ballardie, "A New Approach to Multicast Communication in a Datagram Network", Ph.D. Thesis, University College London, May 1995.

[4] M. Bellare, R. Canetti and H. Krawczyk, "Keying Hash Functions for Message Authentication", *Advances in Cryptology – Crypto '96*, LNCS vol. 1109, Springer-Verlag, 1996.

[5] M. Blum and S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM J. on Comp.*, 13, 1984, 850–864.

[6] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung, "Perfectly-secure key distribution for dynamic conferences, *Advances in Cryptology – Crypto '92*, LNCS vol. 740, Springer-Verlag, 1993, 471–486.

[7] R. Canetti, R. Gennaro, D. Herzberg and D. Naor, "Proactive Security: Long-term protection against break-ins", CryptoBytes, Vol. 3, No. 1, Sprint 1997.

[8] R. Canetti and B. Pinkas, "A taxonomy of multicast security issues", internet draft `draft-canetti-secure-multicast-taxonomy-01.txt`, November 1998.

[9] Y. Desmedt and Y. Frankel, "Threshold cryptosystems", *Advances in Cryptology – Crypto '89*, LNCS vol. 435, Springer-Verlag, 1990, pp. 307–315.

[10] Y. Desmedt, Y. Frankel and M. Yung, "Multi-receiver/Multi-sender network security: efficient authenticated multicast/feedback", *IEEE Infocom '92*, pp. 2045–2054.

[11] M. Dyer, T. Fenner, A. Frieze and A. Thomason, "On Key Storage in Secure Networks", *J. of Cryptology*, Vol. 8, 1995, 189–200.

[12] A. Fiat and M. Naor, "Broadcast Encryption", *Advances in Cryptology – CRYPTO '92*, LNCS vol. 839, 1994, pp. 257–270.

[13] Gemmel P., "An introduction to threshold cryptography", in *CryptoBytes*, Winter 1997, 7–12.

[14] R. Gennaro and P. Rohatgi, "How to sign digital streams", *Advances in Cryptology – CRYPTO '97*, LNCS, vol. 1294, Springer-Verlag, 180–197.

[15] L. Gong and N. Schacham, "Elements of Trusted Multicasting," TR SRI-CSL-94-03, Computer Science Laboratory, SRI International, May 1994.

[16] Harney H. and Muckenhirn C., "Group Key Management Protocol (GKMP) Architecture", *RFC 2094*, July 1997.

[17] *IP Security Protocol (ipsec)*, IETF working group. URL: `http://www.ietf.org/html.charters/ipsec-charter.html`

[18] H. Krawczyk, "SKEME: A Versatile Secure Key Exchange Mechanism for Internet", in *IEEE 1996 Symp. on Network and Dist. Systems Security*.

[19] H. Krawczyk, M. Bellare and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[20] M. Luby, *Pseudorandomness and Cryptographic Applications*, Princeton University Press, 1996.

[21] D. McGrew and A. T. Sherman, "Key Establishment in Large Dynamic Groups Using One-Way Function Trees", manuscript, 1998.

[22] S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting", *Proc. of the ACM SIGCOMM '97*, Cannes, France, Sept. 1997.

[23] M. Naor and O. Reingold, "From Unpredictability to Indistinguishability: A Simple Construction of Pseudo-Random Functions from MACs", *Advances in Cryptology – Crypto '98*, LNCS vol. 1462, Springer-Verlag, 1998, pp. 267–282.

[24] R. L. Rivest, *The MD5 Message Digest Algorithm*, RFC 1321, April 1992.

[25] R. Safavi-Naini and H. Wang, "New results on Multi-receiver Authentication Code," *Advances in Cryptology – EUROCRYPT '98*, LNCS vol. 1403, Springer-Verlag, 1998, pp. 527–541.

[26] The Secure Multicast Group of the Internet Research Task Force, see `http://www.ipmulticast.com/community/smug/`

[27] M. Steiner, G. Tsudik and M. Waidner, "Diffie-Hellman key distribution extended to group communication", *3rd ACM Conf. on Computer and Communications Security*", 1996.

[28] D.M. Wallner, E.J. Harder and R.C. Agee, "Key Management for Multicast: Issues and Architectures". "draft-wallner-key-arch-00.txt".

[29] C.K. Wong, M. Goudaand S. Lam, "Secure Group Communications Using Key Graphs", *Sigcomm '98*.

[30] C.K. Wong and S. Lam, "Digital Signatures for Flows and Multicasts", The University of Texas at Austin, Department of Computer Sciences, Technical Report TR-98-15. July 1998.

Tracing Traitors

Benny Chor[1] and Amos Fiat[2] and Moni Naor[3]

[1] Dept. of Computer Science, Technion, Haifa 32000, Israel.
[2] Dept. of Computer Science, School of Mathematics, Tel Aviv University, Tel Aviv, Israel, and Algorithmic Research Ltd.
[3] Dept. of Computer Science and Applied Math, Weizmann Institute, Rehovot, Israel.

Abstract. We give cryptographic schemes that help trace the source of leaks when sensitive or proprietary data is made available to a large set of parties. This is particularly important for broadcast and database access systems, where the data should be accessible only to authorized users. Such schemes are very relevant in the context of pay television, and easily combine with and complement the Broadcast Encryption schemes of [6].

1 Introduction

If only one person is told about some secret, and this next appears on the evening news, then the guilty party is evident. A more complex situation arises if the set of people that have access to the secret is large. The problem of determining guilt or innocence is (mathematically) insurmountable if all people get the exact same data and one of them behaves treacherously and reveals the secret.

Any data that is to be available to some while it should not be available to others can obviously be protected by encryption. The *data supplier* may give authorized parties cryptographic keys allowing them to decrypt the data. This does not solve the problem above because it does not prevent one of those authorized to view the message (say, Alice) from transferring the *cleartext* message to some unauthorized party (say, Bob). Once this is done then there is no (cryptographic) means to trace the source of the leak. We call all such unauthorized access to data *piracy*. The *traitor* or *traitors* is the (set of) authorized user(s) who allow other, non-authorized parties, to obtain the data. These non-authorized parties are called *pirate users*.

In many interesting cases it is somewhat ineffective piracy if the relevant *cleartext* messages must be transmitted by the "traitor" to the "enemy". Typical cases where this is so include

- Pay-per-view or subscription television broadcasts. It is simply too expensive and risky to start a pirate broadcast station.
- CD ROM distribution of data where a surcharge is charged for different parts of the data. The cleartext data can only be distributed on a similar storage device.
- Online databases, freely accessible (say on the internet) where a charge may be levied for access to all or certain records.

In all these cases, transmitting the cleartext from a traitor, Alice, to an pirate-user, Bob, is either irrelevant or rather expensive. As piracy in all these cases is a criminal commercial enterprise the risk/benefit ratio becomes very unattractive. These three examples can be considered generic examples covering a wide range of data services offered.

Our contribution in this paper may be viewed in the following manner: Consider a ciphertext that may be decrypted by a large set of parties, but each and every party is assigned a different *personal key* used for decrypting the ciphertext. (We use the term personal key rather than private key to avoid confusion with public key terminology). Should the personal key be discovered (by taking apart a television pirate decoder or by counter-espionage), the traitor will be identified.

We note that in fact, our schemes have the very desirable property that the identity of the traitor can be established by considering the pirate decryption process as a black box. It suffices to capture one pirate decoder and it's behavior will identify the traitor, there is no need to "break it open" or read any data stored inside. We use the term pirate decoder to represent the pirate decryption process, this may or may not be a physical box, this may simply be be some code on a computer.

Clearly, a possible solution is to encrypt the data separately under different personal keys. This means that the total length of the ciphertext is at least n times the length of the cleartext, where n is the number of authorized parties. This is clearly impossible in any broadcast environment. This is also very problematic in the context of CD ROM distributed databases because this means that every CD ROM must be different. An encrypted online database, freely accessible as above, must store an individually encrypted copy of the database for each and every authorized user.

The underlying security assumption of our schemes is either information theoretic security (where the length of the personal keys grows with the length of the messages to be transmitted) or it may be based on the security of any symmetric scheme of your choice. In both cases, security depends on a scheme parameter k, the largest group of colliding traitors.

In practice today it is often considered sufficient to prevent piracy by supplying the authorized parties with so-called secure hardware solutions that are designed to prevent interference and access to enclosed cryptographic keys (smartcards and their like). Our schemes do not require any such assumption, they obtain their claimed security without any secure hardware requirements. Should such devices be used to store the keys they will undoubtedly make the attack more expensive, but this is not a requirement.

Fighting piracy in general has the following components:

1. Identify that piracy is going on and prevent the transmittal of information to pirate users, while harming no legitimate users.
2. Take legal measures against the source of such piracy, supply legal evidence of the pirate identity.

Any solution to fighting piracy must be considered in light of the following performance parameters:

(a) What are the memory and computation requirements per authorized user?
(b) What are the memory and computation requirements for the data supplier?
(c) What is the data redundancy overhead? This is measured in multiples of the the cryptographic security parameter and refers to the communications overhead (in broadcast or online systems) or the additional "wasted" storage in CD ROM type systems.

Consider a pirate user who has already obtained all keys required to read a CD ROM in it's entirety. Clearly, there is little one can do technically to prevent her from continuing to use the CD ROM. The situation is somewhat different if the system requires some action on behalf of the data supplier, *e.g.*, television broadcast or online database.

The broadcast encryption scheme of Fiat and Naor [6] deals with disabling active pirate users very efficiently. These schemes allow one to broadcast messages to any dynamic subset of the user set, this is specifically suitable for pay-per-view TV applications but also implies the piracy protection above. These schemes require a single short transmission to disable all pirate decoders if they were manufactured via a collaborative effort of no more than k traitors.

The number of traitors above, k, is a parameter of the broadcast encryption schemes. While this may not be evident at first, the same scheme could be used by any online database supplier to kill off illegitimate access simply by telling users who log on what users are currently blacklisted.

The goal of this paper is to deal *traitor tracing* (item 2 above), *i.e.*, to identify the source of the problem and to deal with it via legal or extra-legal means. Our solution, called traitor tracing, is valid for all examples cited above, broadcast, online, and CD ROM type systems.

We devise *k-resilient traceability* schemes with the following properties:

1. Either the cleartext information itself is continuously transmitted to the enemy by a traitor, or
2. Any captured pirate decoder will correctly identify a traitor and will protect the innocent even if up to k traitors combine and collude.

It would make sense to have both broadcast encryption and traitor tracing schemes available, at different security levels. The costs of such schemes are measured in the memory requirements at the user end and in the total transmission length required. In practice one would want a broadcast encryption scheme with a different security level (measured in the numbers of traitors required to disable the scheme). Fortunately, both types of scheme, at arbitrary security levels, can be trivially combined simply by XOR'ing the results.

We deal with schemes of the following general form: The data supplier generates a base set R of r random keys and assigns subsets of these keys to users, m keys per user (these parameters will be specified later). These m keys jointly form the user personal key. Note that different personal keys may have a nonempty

intersection. We denote the personal key for user u by $P(u)$, this is a set of keys over the base set R.

A traitor tracing message consists of many pairs of (*enabling block*, *cipher block*). The cipher block is the symmetric encryption of the actual data (say a few seconds of a video clip), under some secret random key S. Alternately, it could simply be the XOR of the message with S and we would get an information theoretic secure version of the scheme. The enabling block allows authorized users to obtain S. The enabling block consists of encrypted values under some or all of the r keys at the data supplier. Every user will be able to compute S by decrypting the values for which he has keys and then computing the actual key from these values. The computation on the user end, for all schemes we present, is simply the exclusive or of all values the user has been able to decrypt.

Figures 1 and 2 describe the general nature of our traitor tracing schemes.

Traitors may conspire and give an unauthorized user (or users) a subset of their keys so that the unauthorized user will also be able to compute the real message key from the values he has been able to decrypt. The goal of the system designer is to assign keys to the users such that when a pirate decoder is captured and the keys it possesses are examined, it should be possible to detect at least one traitor, subject to the limitation that the number of traitors of is at most k. (We cannot hope to detect all traitors as one traitor may simply provide his personal key and others may provide nothing).

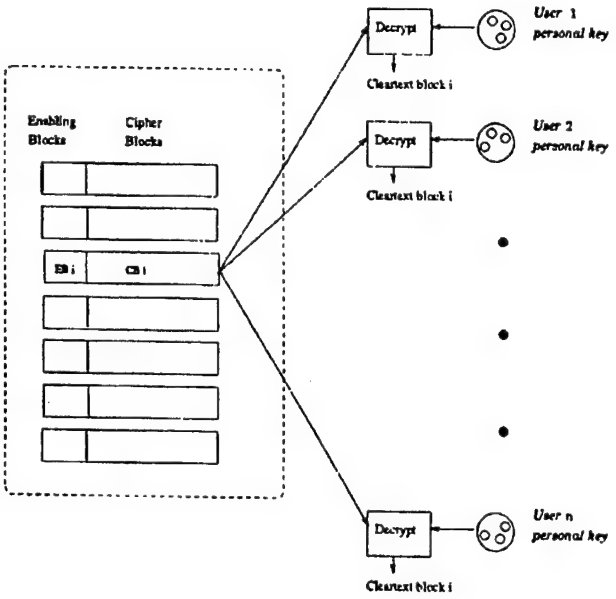


Fig. 1.

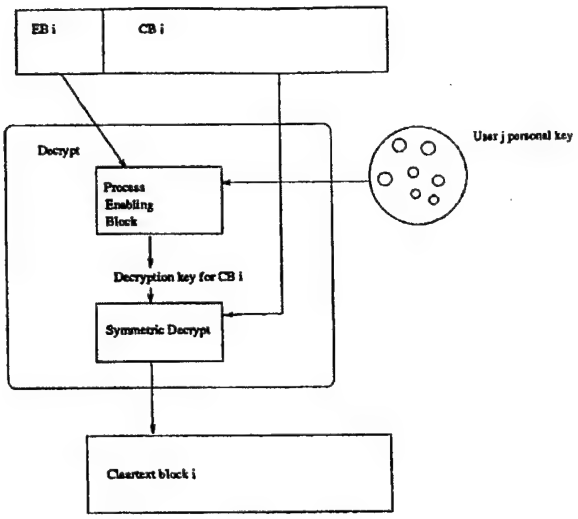


Fig. 2.

We remark that in many cases it is preferable to predetermine a fixed number of users n, and to assign them personal keys, even if the actual number of users is smaller. Later users who join the system by purchasing a subscription to a television station, online database, or CD ROM access privilege are assigned personal keys from those preinstalled. This is especially important in the case of data distributed on CD ROM.

1.1 An Example

Using the 1-level secret scheme described hereinafter, allocating 5% of a compressed MPEG II digital video channel to the traitor tracing scheme allows us to change keys every minute or so (a new enabling block every minute).

The traitor tracing scheme is resilient to $k = 32$ traitors, with probability $1 - 2^{-10}$, and can accommodate up to 1,000,000,000 authorized users. The total number of keys stored by the data supplier (the television broadcaster) is 2^{19}, the personal key of every user consists of 2^{13} keys. These parameters are overly pessimistic because they are derived from the general theorem concerning the scheme using the Chernoff bound.

In practice, there is no real need to change keys every minute, even changing keys once every hour will make any pirate broadcaster give up in despair.

2 Definitions

For messages generated by a data supplier for a set of n users, we define three elements that jointly constitute a *traceability scheme*:

- A *user initialization scheme*, used by the data supplier to add new users. The data supplier supplies user u_i with her personal key, in our case this consists of a set $P(u_i)$ containing decryption keys.
- A *decryption scheme*, used by every user to decrypt messages generated by the data supplier. In our schemes, the messages are decrypted block by block where every block decryption consists of a preliminary decryption of encrypted keys in the enabling block, combining the results to obtain a common key, followed by a decryption of the cipher block.
- A *traitor tracing algorithm*, used upon confiscation of a pirate decoder, to determine the identity of a traitor. We assume below that the contents of a pirate decoder can be viewed by the traitor tracing algorithm.

We distinguish between circumstances where the decryption schemes used by all users are in the public domain, whereas the decryption keys themselves are kept secret, called *open schemes*, versus the case where the actual decryption scheme as well as the keys are kept secret, called *secret schemes*.

The goal of an adversary is to construct a pirate decoder that allows decryption and prevents the guilty from being identified. In particular, one way to ensure that the guilty are safe is to try to incriminate someone else. Clearly, the adversaries task is no harder with an open scheme compared to a secret scheme. On the other hand, secret schemes pose additional security requirements at the data supplier cite and the correctness of the traitor identity may be based on probabilistic arguments, which may be somewhat less convincing in a court of law.

We present efficient schemes of both types, and our constructions give better results for secret schemes. It is clearly advantageous to use secret schemes in practice, and any real implementation will do so.

To simplify the definitions below we will assume that it is impossible to guess a secret key. The probability of guessing a secret key is exponentially small in the length of the key, and thus we will ignore this question in the definitions below. An alternative would be to talk about probability differences rather than absolute probabilities, this is done in [6] and analogous definitions could be used here.

Definition 1. An n user open traceability scheme is called k*-resilient* if for every coalition of at most k traitors the following holds: Suppose the coalition uses the information its members got in the initialization phase to construct a pirate decoder. If this decoder is capable of applying the decryption scheme, then the traitor tracing algorithm will correctly identify one of the coalition members.

Definition 2. An n user secret traceability scheme is called (p, k)*-resilient* if for all but at most p of the $\binom{n}{k}$ coalition of k traitors the following holds: Suppose the coalition uses the information its members got in the initialization phase to construct a pirate decoder. If this decoder is capable of applying the decryption scheme, then the traitor tracing algorithm will identify one of the coalition members with probability at least $1 - p$.

3 Construction of Traceability Schemes

In this section we describe three constructions of k-resilient traceability schemes. All these schemes are based on the use of hash functions, combined with *any* private key cryptosystem. (For more information on hash functions and their applications, see [7, 3, 9, 5].) The basic use of hash functions is to assign decryption keys to authorized users in a manner which prevents any coalition of traitors from combining keys taken from the personal keys of its members into a set of keys that allows decryption yet is "close" to the personal key of any innocent user.

The first scheme is the simplest one. It is an open scheme, based on "one level" hash functions. Each hash function maps the n users into a set of $2k^2$ decryption keys. The keys themselves are kept secret, but the mapping (which user is mapped to what key) is publicly known. This is a simple scheme, but its performance can be improved upon: Every user personal key consists of $O(k^2 \log n)$ decryption keys, and the enabling block consists of $O(k^4 \log n)$ encrypted keys.

The second scheme is an open "two level" scheme. Here, a set of first level hash functions map the n users into a set of size k. each function thereby induces a partition of the n users to k subsets. Each of these subsets is mapped separately by "second level" hash functions into $\log^2 k$ decryption keys. This scheme requires $O(k^2 \log^2 k \log n)$ keys per user, and an enabling block of $O(k^3 \log^4 k \log n)$ encrypted keys.

The third scheme is a "one level" secret scheme. Here, we assume that the hash functions, as well as the decryption keys, are kept secret. There is a positive probability p $(0 < p < 1)$ that the adversary will be able to produce pirate decoders which prevent the identification of any traitor.

However, even if the keys known to the k collaborators enable the construction of such "wrongly incriminating" pirate decoders, choosing such set is highly improbable. Even if this unlikely event occurs, the adversary will not know that this is the case.

Being a secret scheme implies that the adversary does not know what keys corresponds to any specific user. The personal key consists of $O(k \log(n/p))$ decryption keys, and has $O(k^2 \log(n/p))$ encrypted keys per enabling block.

All schemes are constructed by choosing hash values at random, and using probabilistic arguments to assert that the desired properties hold with overwhelming probability. Therefore, these schemes are not constructive. However, the properties of the simplest scheme can be verified.

3.1 A Simple Scheme

Let k be an upper bound on the number of traitors. Every enabling block consists of r encryptions, and m denotes the number of keys comprising a user personal key.

We first deal with the case $k = 1$. The data supplier generates $r = 2 \log n$ keys $s_1^0, s_1^1, s_2^0, s_2^1, \ldots, s_{\log n}^0, s_{\log n}^1$. The personal key for user i is the set of $m = \log n$ keys $s_1^{b_1}, s_2^{b_2}, \ldots, s_{\log n}^{b_{\log n}}$, where b_i is the ith bit in the ID of u.

To encrypt a secret s, the data supplier splits it into $\log n$ secrets $s_1, s_2, \ldots, s_{\log n}$, i.e., the data supplier chooses random $s_1, s_2, \ldots, s_{\log n}$ such that q is the bitwise XOR of the s_i's (its j-th bit equals $s^{(j)} = XOR_{i=1}^{\ell} s_i^{(j)}$). The value s_i is encrypted using keys s_i^0 and s_i^1 and both encryptions are added to the enabling block. Every user u can reconstruct all the s_i's and hence can decrypt s. On the other hand, any pirate decoder must contain a key for every $1 \leq i \leq \log n$ (otherwise s_i would remain unknown and consequently s could not be obtained).

Thus, given that at most one traitor is involved, the keys stored in the pirate decoder uniquely identify the traitor.

When dealing with larger coalition, the idea is to generalize the above scheme. Instead of one bit per index we will have larger domains (and have a key for every element in the domain). We will also split s into more than $\log n$ parts and have appropriately more indices or hash functions. The major difficulty we encounter is in the procedure for detecting traitors. Since, unlike the case $k = 1$, keys may be mixed from several members of the coalition, we must make sure that the two users are not only different on some indices, but are different in almost all indices. A detailed description of the scheme is given below.

Initialization: A set of ℓ "first level" hash functions $h_1, h_2, \ldots, h_\ell$ is chosen at random by the data supplier. Each hash function h_i maps $\{1, \ldots, n\}$ into an independent set of $2k^2$ random keys, $S_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,2k^2}\}$. The personal key for user u is the set $P(u) = \ell$ keys $\{h_1(u), h_2(u), \ldots, h_\ell(u)\}$.

Distributing a Key: For each i ($i = 1, 2, \ldots, \ell$) the data supplier encrypts a key s_i under each of the $2k^2$ keys in S_i. The final key s is the bitwise XOR of the s_i's (its j-th bit equals $s^{(j)} = XOR_{i=1}^{\ell} s_i^{(j)}$). Each authorized user has one key from S_i, so he can decrypt every s_i, and thus compute s.

Parameters: The memory required per user is $m = \ell$ keys. An enabling block to encode a secret value s consists of $= 2k^2\ell$ key encryptions.

Fraud: The k traitors can get together and combine their personal keys. They may choose one key from every set S_i ($i = 1, 2, \ldots, \ell$). These ℓ keys are put together in a pirate decoder. This set of keys F, enables the purchaser of such a decoder to decrypt every s_i, and thus compute s.

Detection of Traitors: Upon confiscation of a pirate decoder, the set of keys in it, F, is exposed. The set F must contain at least ℓ keys (at least one key per set S_i). Denote the key with minimum index in S_i by $f_i \in S_i$. For each i, the users in $h_i^{-1}(f_i)$ are identified and marked. The user with largest number of marks is exposed.

Goal: We want to show that for all coalitions of size k, the probability of exposing a user who is not a traitor is negligible.

Clearly, at least one of the traitors contributes at least ℓ/k of the keys to the pirate decoder (we ignore duplicate keys from the same S_i). We want to show that the probability (over all choices of hash functions) that a good user is marked ℓ/k times is negligible. Consider a specific user, say 1, and a specific coalition T of k traitors (which does not include 1). As hash functions are chosen at random, the value $a_i = f_i(1)$ is uniformly distributed in S_i. The coalition gets at most k keys in S_i. The probability that a_i is among these keys is at most $1/2k$.

Let X_i be a zero-one random variable, where $X_i = 1$ if $a_i \in h_i(T)$. The mean value of $\sum_{i=1}^{\ell} X_i$ is $\ell/2k$. We use the following version of Chernoff bound (see [2], Theorem A.12) to bound the probability that $\sum_{i=1}^{\ell} X_i \geq \ell/k$. Let $X_1, \ldots, X_\ell$ be mutually independent random variables, with

$$Pr(X_j = 1) = p$$
$$Pr(X_j = 0) = 1 - p$$

Then, for all $\beta \geq 1$

$$Pr\left(\frac{1}{\ell}\sum_{j=1}^{\ell} X_j \geq \beta p\right) < \left(\frac{e^{\beta-1}}{\beta^\beta}\right)^{p\ell}.$$

In our case, substituting $p = 1/2k$ and $\beta = 2$, we have

$$Pr\left(\frac{1}{\ell}\sum_{i=1}^{\ell} X_i \geq \frac{1}{k}\right) < \left(\frac{e}{4}\right)^{\ell/2k} < 2^{-\ell/4k}.$$

In order to overcome all $\binom{n}{k}$ coalitions and all n choices of users, we choose ℓ satisfying

$$n \cdot \binom{n}{k} \cdot 2^{-\ell/4k} < 1,$$

namely $\ell > 4k^2 \log n$. With this parameter, there is a choice of ℓ hash functions such that for every coalition and every authorized user not in the coalition, the user is not incriminated by the tracing algorithm. We summarize these results in the next theorem:

Theorem 3. *There is an open k-resilient traceability scheme, where a user personal key consists of $m = 4k^2 \log n$ decryption keys, and an enabling block consists of $r = 8k^4 \log n$ key encryptions.*

Explicit Constructions: The discussion above shows the existence of open k resilient traceability schemes, and does provide us with a randomized method for constructing the scheme that works with high probability. It does not, however, suggests an explicit construction. Note however that a given construction can be verified quite efficiently. The idea is to examine all the pairs of elements u, v and check the number of function h_i such that $h_i(v) = h_j(u)$. If this number is smaller than ℓ/k^2 than we can conclude that no coalition T of at most k elements "covers" more than a $1/k$ fraction of the keys of u and hence cannot incriminate u.

By considering pairwise differences, we can phrase the construction problem as a problem in coding theory (see [8] for more information): construct a code with n codewords over an alphabet of size $2k^2$ of length ℓ such that the distance between every two codewords is at least $\ell - \ell/k^2$. The goal is construct such a code with as small ℓ as possible. There are no known explicit construction that match the probabilistic bound. For the best known construction see [1] and references therein. For small k the constructions there yields a scheme with $m \in O(k^6 \log n)$ and $r \in O(k^8 \log n)$.

3.2 An Open Two Level Scheme

The "two level" traceability scheme, described in this subsection, more complicated than the simple scheme, but it saves about a factor of k in the broadcast overhead.

Theorem 4. *There is an open k-resilient traceability scheme, where a user personal key consists of $m = 2k^2 \log^2 k \log n$ decryption keys, and an enabling block consists of $r = 4k^3 \log^4 k \log n$ key encryptions.*

Proof. We describe the system, step by step. As in the one-level scheme, the proof is existential. We do not know however how to verify efficiently that a given scheme is "good".

Initialization: A set of ℓ "first level" hash functions $h_1, h_2, \ldots, h_\ell$, each mapping $\{1, \ldots, n\}$ to $\{1, \ldots, k\}$, is chosen at random. For each i ($i = 1, 2, \ldots, \ell$) and each element a in $\{1, \ldots, k\}$, a set of d "second level" hash functions $g_{i,a,1}, \ldots, g_{i,a,d}$ is chosen at random. Each second level function $g_{i,a,j}$ maps the users in $h_i^{-1}(a) \subset \{1, \ldots, n\}$ into a set of $4 \log^2 k$ random independent keys (the ranges of different functions are independent).

Each user $u \in \{1, \ldots, n\}$ receives $\ell \cdot d$ keys

$$\begin{array}{ccc} g_{1,h_1(u),1}(u), & \cdots & , g_{1,h_1(u),d}(u) \\ \vdots & & \vdots \\ g_{\ell,h_\ell(u),1}(u) & , \ldots , & g_{\ell,h_\ell(u),d}(u) \end{array}$$

Distributing a Key: The data supplier chooses at random ℓ independent keys $s_1, \ldots, s_\ell$. The final key is $s = BITWISE - XOR_{i=1}^{\ell}(s_i)$.

For each i ($i = 1, 2, \ldots, \ell$), a ($a = 1, \ldots, k$), and j ($j = 1, \ldots, d$), let $s_{i,a,j}$ be an independent random key, satisfying

$$\begin{aligned} s_i &= BITWISE - XOR(s_{i,1,1}, \ldots, s_{i,1,d}) \\ &= BITWISE - XOR(s_{i,2,1}, \ldots, s_{i,2,d}) \\ &\vdots \\ &= BITWISE - XOR(s_{i,k,1}, \ldots, s_{i,k,d}) \end{aligned}$$

The key $s_{i,a,j}$, encrypted under each of the $4 \log^2 k$ keys in the range of the function $g_{i,a,j}$, is added to the enabling block.

User u possesses the d keys $g_{i,h_i(u),1}(u), \ldots, g_{i,h_i(u),d}(u)$ and so he is capable of decoding $s_{i,h_i(u),1}, \ldots, s_{i,h_i(u),d}$, allowing him to reconstruct s_i and then compute the final key s.

Parameters: The personal key consists of $m = \ell d$ keys. The total number of key encryptions in an enabling block encoding s is $4k\ell d \log^2 k$.

Fraud: The k traitors can get together and expose their own keys in order to construct a pirate decoder. By the bit sensitivity of XOR, the box must be able to decrypt every s_i ($i = 1, 2, \ldots, \ell$). To do this, the decoder must be

able to decrypt a complete row $s_{i,a,1}, \ldots, s_{i,a,d}$ for some a, $1 \leq a \leq k$. So, for each i $(i = 1, 2, \ldots, \ell)$ the traitors choose $a = h_i(u)$ for some $u \in T$, and d keys $g_{i,a,1}(u_1), \ldots, g_{i,a,d}(u_d)$, where $u_1, \ldots, u_d \in T$ and $h_i(u_1) = h_i(u_2) = \ldots = h_i(u_d) = a$. For every i, these d keys are placed in the pirate decoder.

Detection of Traitors: Upon confiscation of a pirate decoder, the set of keys in it, F, is exposed. As argued above, the decoder must contain a block of d keys of the form $k_{i,a,1} = g_{i,a,1}(u_1), \ldots, k_{i,a,d} = g_{i,a,d}(u_d)$ corresponding to each i $(i = 1, 2, \ldots, \ell)$. (If more than one row is in the decoder, only the one with minimum a is used by the detection algorithm.) For each $j = 1, \ldots, d$, the detective identifies the users in $g_{i,a,j}^{-1}(k_{i,a,j})$. Each of these users is called *marked*. All users who are marked at least $d/\log k$ times, are suspects for s_i. The user who is a suspect for the largest number of s_i's is identified as a traitor.

Goal: We want to show that there is a choice of hash functions such that for all coalitions, a good user is never identified as a traitor.

Consider a specific user, say 1, and a specific coalition T of k traitors (which does not include 1). We first bound the probability that user 1 will be a suspect for s_i. The first level hash function h_i partitions the users to k subsets $\{h_i^{-1}(1), \ldots, h^{-1}(k)\}$. The expected maximum number of traitors in these k subsets is $\log k / \log\log k$. The probability that user 1 is hashed to a subset together more than $\log k$ traitors is at most $1/16k$ [2]. Denote $h_i(1) = a$. Consider the conditional probability space where $T \cap h_i^{-1}(a)$ contains indeed at most $\log k$ traitors. In this conditional space, the d keys $k_{i,a,1}, \ldots, k_{i,a,d}$ in the pirate decoder come from the personal keys of $T \cap h_i^{-1}(a)$. As this set contains fewer than $\log k$ members, there must be at least one member in $T \cap h_i^{-1}(a)$ who is marked at least $d/\log k$ times. Therefore at least one member of T is a suspect for s_i.

Returning to our innocent user 1, the detective marks user 1 with respect to $g_{i,a,j}$ if there is some $u \in T \cap h_i^{-1}(a)$ such that $g_{i,a,j}(1) = g_{i,a,j}(u)$. The range of $g_{i,a,j}$ contains $4\log^2 k$ keys. At most $\log k$ of these are in $g_{i,a,j}\left(T \cap h_i^{-1}(a)\right)$. So the probability that user 1 is marked with respect to $g_{i,a,j}$ is at most $1/(4\log k)$. The expected number of times user 1 will be marked, with respect to the d functions $g_{i,a,1}, \ldots, g_{i,a,d}$, is $d/(4\log k)$. We use the Chernoff bound to estimate the probability that user 1 is a suspect for s_i.

Set $X_j = 1$ if user 1 is marked with respect to $g_{i,a,d}$, and $X_j = 0$ otherwise. Then $Pr(X_j = 1) \leq 1/(4\log k)$. By the version of the Chernoff bound mentioned above, with $p = 1/(4\log k)$ and $\beta = 4$,

$$Pr\left(\frac{1}{d}\sum_{j=1}^{d} X_j \geq \frac{1}{\log k}\right) < \left(\frac{e^3}{4^4}\right)^{d/4\log k} \leq 2^{-3d/4\log k}.$$

Setting $d = 2\log^2 k$, the conditional probability that user 1 is a suspect for s_i is at most $2^{-3\log k/2} < 1/16k$. The probability of the condition not happening is at most $1/16k$. So overall, the total (unconditional) probability that user 1 is the suspect for s_i is at most $1/8k$.

For $i = 1, \ldots, \ell$, let $Y_i = 1$ if 1 is the suspect for s_i, and $Y_i = 0$ otherwise. Then

$$Pr\left(\frac{1}{\ell}\sum_{i=1}^{\ell} X_i \geq \frac{1}{2k}\right) < \left(\frac{e^7}{8^8}\right)^{\ell/8k} < 2^{-\ell/k}.$$

So with probability at least $1 - 2^{-\ell/k}$, user 1 is a suspect for fewer than ℓ/k of the s_i.

For every s_i $(i = 1, \ldots, \ell)$, at least one member of T is a suspect for s_i. T contains k traitors, and so there must be one or more traitor who is a suspect for at least ℓ/k s_i's. Therefore the probability that user 1 is mistakenly identified as a traitor is smaller than $2^{-\ell/k}$. The probability that for one of the $\binom{n}{k}$ possible coalitions T of size k, some good user is mistakenly identified, is smaller than $n \cdot \binom{n}{k} \cdot 2^{-\ell/k}$. Setting $\ell = k^2 \log n$, this probability is smaller than 1. This means that there exists a choice of hash functions h_i and $g_{i,a,j}$ such that a good user is never mistakenly identified as a traitor. The resulting open k-traceability scheme has parameters $m = \ell d = 2k^2 \log^2 k \log n$ and $r = 2k\ell d \log^2 k = 4k^3 \log^4 k \log n$.

3.3 A Secret One Level Scheme

We simplify the construction and improve its costs by using a secret scheme. The proposed scheme is one level, and the hash values of users are kept secret. The major source of saving is that it suffices to map the n users into a set of $4k$ keys (rather than k^2 keys as in the simple one level scheme). A coalition of size k will contain the key of any specific user with constant probability. However, as the traitors do not know which key this is, any key they choose to insert into the pirate decoder will miss the key of the authorized user (with high probability).

Initialization: Each user u $(u \in \{1, 2, \ldots, n\})$ is assigned a random name n_u from a universe $\mathcal{U}$ of size exponential in n. These names are kept secret. A set of ℓ hash functions $h_1, h_2, \ldots, h_\ell$ are chosen independently at random. Each hash function h_i maps $\mathcal{U}$ into a set of $4k$ random keys $S_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,4k}\}$. The hash functions are kept secret as well. User u receives, upon initialization, ℓ keys $\{h_1(n_u), h_2(n_u), \ldots, h_\ell(n_u)\}$.

Distributing a Key: For each i $(i = 1, 2, \ldots, \ell)$ the data supplier encrypts a key s_i under each of the $4k$ keys in S_i. The final key is the bitwise XOR of the s_i's (the j-th bit is $s^{(j)} = XOR_{i=1}^{\ell} s_i^{(j)}$). Each authorized user has one key from S_i, so he can decrypt every s_i, and thus compute s.

Parameters: The memory required per user is $m = \ell$ keys. The total number of broadcasts, used in distributing the key s, is $r = 4k\ell$.

Fraud: The k traitors can get together and expose their own keys. Given these keys, they chose one key per set S_i $(i = 1, 2, \ldots, \ell)$. These ℓ keys are put together in a pirate decoder. This set of keys F, enables the purchaser of such decoder to decrypt every s_i, and thus compute s.

Detection of Traitors: Upon confiscation of a pirate decoder, the set of keys in it, F', is exposed. F' contains ℓ keys, one per set S_i. Denote these keys by $f_i \in S_i$. For each i, the users in $h_i^{-1}(f_i)$ are identified and marked. The user with largest number of marks is exposed.

Goal: We want to show that for all (almost all) coalitions, the probability of exposing a user who is not a traitor is negligible.

Clearly, at least one of the traitors contributes at least ℓ/k of the keys to the pirate decoder. We want to show that the probability that a good user is marked ℓ/k times is negligible. Consider a specific user, say 1, and a specific coalition T of k traitors (which does not include 1). As the name assigned to user 1 is random and the hash functions are random, the value $a_i = h_i(n_1)$ is uniformly distributed in S_i, even given the k values hashed by h_i from the names of the coalition members. The probability that the value f_i, chosen by the coalition to the pirate decoder, equals a_i is therefore $q = 1/4k$. Let X_i be a zero-one random variable, where $X_i = 1$ if $a_i = f_i$. The mean value of $\sum_{i=1}^{\ell} X_i$ is $\ell/4k$. By the Chernoff bound

$$Pr\left(\frac{1}{\ell}\sum_{i=1}^{\ell} X_i \geq 4q\right) < \left(\frac{e^3}{4^4}\right)^{q\ell} < 2^{-3\ell/4k}.$$

In order to overcome all but p of the $\binom{n}{k}$ coalitions and all n choices of users, we choose ℓ satisfying $n \cdot 2^{-3\ell/4k} < p$. That is, $4k\log(n/p)/3 < \ell$, which gives

Theorem 5. *There is a (p,k)-resilient secret traceability scheme, where a user personal key consists of $m = 4k\log(n/p)/3$ decryption keys, and an enabling block consists of $16k^2\log(n/p)/3$ key encryptions.*

4 Lower Bounds

In this section we derive lower bounds for the case where incrimination has to be absolute, i.e. with no error probability. We assume that the keys the data supplier distributes to the users are unforgeable. This is not accurate, since there is always the small chance that the adversary guesses the keys of the user it wants to incriminate. However, we distinguish between the probability of guessing the keys (which is exponentially small in the length of the key) and the probability of incrimination for other reasons which we would like to be zero. Our view of the system is therefore as follows: let the set of keys used be $S = \{s_1, s_2, \ldots s_r\}$ and let each user i obtain a subset $U_i \subset S$ of size m.

Claim 1 *If no coalition of k users $i_1, i_2, \ldots i_k$ should be able to incriminate a user $i_0 \notin \{i_1, i_2, \ldots i_k\}$, then for all such $i_0, i_1, i_2, \ldots i_k$ we should have that*

$$U_{i_0} \not\subset \cup_{j=1}^{k} U_{i_j}$$

Proof. Suppose not, i.e. there exist $i_0, i_1, \ldots i_k$ such that $U_{i_0} \subset \cup_{j=1}^{k} U_{i_j}$, then the coalition of $i_1, i_2, \ldots i_k$ can reconstruct the keys of U_{i_0} and put them in the pirate decoder for sale. Anyone examining the contents of the box will have to deduce that i_0 is the traitor that generated it.

Luckily, the issue of set systems obeying the conditions of Claim 1 has been investigated by Erdös, Frankl and Füredi [4]. From Theorem 3.3 and Proposition 3.4 there we can deduce that r is $\Omega(\min\{n, k^2\frac{\log n}{\log\log n}\})$ and from Proposition 2.1 there we get that $m \geq k\frac{\log n}{\log r}$. Hence we have:

Theorem 6. *In any open k-resilient traceability scheme distributing every one of the n user m keys out of r we have that r is $\Omega(\min\{n, k^2\frac{\log n}{\log\log n}\})$ and $m \geq k\frac{\log n}{\log r}$.*

Note that the lower bounds on both r and m are roughly a factor of k smaller than the best construction we have for an open traceability system.

Acknowledgments

We are very grateful to Alain Catrevaux, Christophe Declerck, and Henri Joubaud for educating us on the issues of pay television and for motivating all of this work. We also thank to Amos Beimel for his comments on earlier drafts of this manuscript.

References

1. N. Alon, J. Bruck, J. Naor, M. Naor and R. Roth, *Construction of Asymptotically Good Low-Rate Error-Correcting Codes through Pseudo-Random Graphs*, IEEE Transactions on Information Theory, vol. 38 (1992), pp. 509-516.
2. N. Alon and J. Spencer, **The Probabilistic Method**, Wiley, 1992.
3. J. L. Carter and M. N. Wegman, *Universal Classes of Hash Functions*, Journal of Computer and System Sciences 18 (1979), pp. 143-154.
4. P. Erdös, P. Frankl, Z. Füredi, *Families of finite sets in which no set is covered by the union of r others*, Israel J. of math. **51**, 1985, pp. 79–89.
5. M.L. Fredman, J. Komlós and E. Szemerédi, *Storing a Sparse Table with $O(1)$ Worst Case Access Time*, Journal of the ACM, Vol 31, 1984, pp. 538–544.
6. A. Fiat and N. Naor, *Broadcast Encryption*, Proc. Advances in Cryptology - Crypto '93, 1994, pp. 480–491.
7. K. Mehlhorn, **Data Structures and Algorithms: Sorting and Searching**, Springer-Verlag, Berlin Heidelberg, 1984.
8. F. J. MacWilliams and N. J. A. Sloane, **The theory of error correcting codes**, North Holland, Amsterdam, 1977.
9. M. N. Wegman and J. L. Carter, *New Hash Functions and Their Use in Authentication and Set Equality*, Journal of Computer and System Sciences 22, pp. 265-279 (1981).

An Efficient Public Key Traitor Tracing Scheme

Dan Boneh
dabo@cs.stanford.edu

Matthew Franklin
franklin@cs.ucdavis.edu

Abstract

We construct a public key encryption scheme in which there is one public encryption key, but *many* private decryption keys. If some digital content (e.g., a music clip) is encrypted using the public key and distributed through a broadcast channel, then each legitimate user can decrypt using its own private key. Furthermore, if a coalition of users collude to create a new decryption key then there is an efficient algorithm to trace the new key to its creators. Hence, our system provides a simple and efficient solution to the "traitor tracing problem". A minor modification to the scheme enables it to resist an adaptive chosen ciphertext attack. Our techniques apply error correcting codes to the discrete log representation problem.

1 Introduction

Consider the distribution of digital content to subscribers over a broadcast channel. Typically, the distributor gives each authorized subscriber a hardware or software decoder ("box") containing a secret decryption key. The distributor then broadcasts an encrypted version of the digital content. Authorized subscribers are able to decrypt and make use of the content. This scenario comes up in the context of pay-per-view television, and more commonly in web based electronic commerce (e.g. broadcast of online stock quotes or broadcast of proprietary market analysis).

However, nothing prevents a legitimate subscriber from giving a copy of her decryption software to someone else. Worse, she might try to expose the secret key buried in her decryption box and make copies of the key freely available. The "traitor" would thus make all of the distributor's broadcasts freely available to non-subscribers. Chor, Fiat and Naor [5] introduced the concept of a *traitor tracing scheme* to discourage subscribers from giving away their keys. Their approach is to give each subscriber a distinct set of keys that both identify the subscriber and enable her to decrypt. In a sense, each set of keys is a "watermark" that traces back to the owner of a particular decryption box. A coalition of traitors might try to mix keys from many boxes, to create a new pirate box that can still decrypt but cannot be traced back to them. A traitor tracing scheme is "k-collusion resistant" if at least one traitor can always be identified when k of them try to cheat in this way. In practice, especially with tamper-resistant decryption boxes, it may suffice for k to be a fairly small integer, e.g., on the order of 20.

In this paper we present an efficient public key traitor tracing scheme. The public key settings enable anyone to broadcast encrypted information to the group of legitimate receivers. Previous solutions were combinatorial with probabilistic tracing [5, 11, 15, 16, 17], and could be either public-key or symmetric-key. Our approach is *algebraic* with *deterministic* tracing, and is inherently public-key. Our approach is much more efficient than the public-key instantiations of previous combinatorial constructions. One earlier construction was algebraic in nature [9], although it was later determined to be insecure [18].

Previous approaches [5, 11] incur an overhead that is proportional to the logarithm of the size of the population of honest users. In a commercial setting such as a web broadcast or pay-per-view tv, where the number of subscribers might be in the millions, this is a significant factor. Our approach eliminates this factor. Furthermore, secret keys in our scheme are very short. Each private key is just the discrete log of a single element of a finite field (e.g., as small as 160 bits in practice). The size of an encrypted message is just $2k+1$ elements of the finite field. The work required to encrypt is about $2k+1$ exponentiations. Decryption takes far less than $2k+1$ exponentiations. During decryption, only the final exponentiation uses the private key, which can be helpful when the secret is stored on a weak computational device.

Previous probabilistic tracing methods try to maximize the chance of catching just *one* of the traitors while minimizing the chance of accusing an innocent user. Our tracing method is deterministic. Innocent users are never accused, as long as the number of colluders is at or below the collusion threshold, and as long as the complexity assumption remains true. Even when more than k (but less than $2k$) traitors collude, some information about the traitors can be recovered.

The intuition behind our system is as follows. Each private key is a different solution vector for the discrete log representation problem with respect to a fixed base of field elements. We can show that the pirate is limited to forming new keys by taking convex combinations of stolen keys. If every set of $2k$ keys is linearly independent, then every convex combination of k keys can be traced uniquely (but not necessarily efficiently). By deriving our keys from a Reed-Solomon code in the appropriate way, we can take advantage of efficient error correction methods to trace uniquely and efficiently. We note that the multi-dimensional discrete log representation problem has been previously used, e.g., for incremental hashing [1] and Signets [7].

Our scheme is traceable if the discrete log problem is hard. The encryption scheme is secure (semantic security against a passive adversary) if the decision Diffie-Hellman problem is hard. A small modification yields security against an adaptive chosen ciphertext attack under the same hardness assumption. That level of protection can be important in distribution scenarios where the decryption boxes (or decryption software) are widely deployed and largely unsupervised.

In Section 2 we give definitions for the traitor tracing problem. Our basic scheme and a non-black-box tracing algorithm is described in Section 3. Black box tracing against an arbitrary pirate is detailed in Section 4. Black box tracing against a restrictive "single-key pirate" is given in Section 5. Chosen ciphertext security is considered in Section 6, and an open problem is discussed in Section 7. Conclusions are given in Section 8, including an application of our scheme to defending against software piracy.

2 Definitions

For a detailed presentation of the traitor tracing model, see [5]. A public key *traitor tracing* encryption scheme is a public key encryption system in which there is a unique encryption key and multiple decryption keys. The scheme is made up of four components:

Key Generation: The key generation algorithm takes as input a security parameter s, a number ℓ of private keys to generate, and a number k which we call the collusion bound. It outputs a public encryption key e and a list of distinct private decryption keys $d_1, \ldots, d_\ell$. Any decryption key can be used to decrypt a ciphertext created using the encryption key.

Encryption: The encryption algorithm takes a public encryption key e and a message M and outputs

a ciphertext C.

Decryption: The decryption algorithm takes a ciphertext C and any of the decryption keys d_i and outputs the message M. This is an "open" scheme in the sense that only the short decryption keys are secret while the decryption method can be public.

The pirate: The pirate is a party (an algorithm) that is given the public key and k random decryption keys $d_1, \ldots, d_k$ from the set of all ℓ keys. Using the public key and the k private keys, the pirate creates a pirate decryption box (or decryption software) $\mathcal{D}$. The pirate box $\mathcal{D}$ must correctly decrypt all but a negligible fraction of the valid ciphertexts generated by the encryption algorithm.

Tracing: The encryption scheme is said to be "k-resilient" if there is a *tracing algorithm* that, given $\mathcal{D}$ created by some pirate, and all random bits used during key generation, determines at least one of the d_i's in the pirate's possession used to create $\mathcal{D}$. The tracing algorithm is said to be "black box" if its only use of $\mathcal{D}$ is as an oracle to query on various inputs.

Clearly black box tracing is preferable to tracing algorithms that rely on information embedded in the implementation of $\mathcal{D}$. After all, extracting information from the implementation of $\mathcal{D}$ might be difficult: the software executable might be obfuscated, or $\mathcal{D}$ might be implemented in a temper resistant device. Black box tracing is also important for proving that we do not assume the pirate must embed specific information in the pirate decoder.

We describe black box tracing in more detail. One notion of black box tracing is that the tracer uses the pirate decoder as a decryption oracle. Given a string C the pirate decoder outputs either (1) "`invalid`" meaning C is not a valid ciphertext, or (2) outputs a plaintext M which it claims is the decryption of C. In this model the tracer sees the full output of the pirate decoder, namely the decryption of C. In some settings such full access to the pirate decoder may not be possible. For example, consider a pirate decoder that's intended to play music. The decoder is given an encrypted music clip and either plays the music, or says that the given music clip is invalid. When the tracer queries the pirate decoder its only feedback is whether the decoder plays the music or outputs an error. In other words, all the tracer learns is whether the decoder was successful in decrypting the given ciphertext or not. The tracer does not see the full output of the pirate decryption algorithm. Hence, we can define two types of black box tracing:

Full access black box tracing: In this model the tracer issues a query C to the pirate decoder. The decoder returns either (1) a plaintext M which is supposedly the decryption of C, or (2) it returns `invalid` meaning the ciphertext C is invalid. The decoder can behave maliciously and return either one of the above for any query. When the decoder returns a message M it is free to choose the message maliciously. However, if C is a well formed ciphertext the decoder must return the decryption of C.

Minimal access black box tracing: In this model the queries to the pirate decoder are a pair $\langle C, M \rangle$ where C is some ciphertext query and M is some plaintext. The pirate decoder returns either (1) `valid` meaning C is a valid encryption of M, or (2) `invalid` meaning C is not. The decoder can behave maliciously and return either one of the above for any query. However, if C is a well formed ciphertext which is an encryption of M the pirate decoder must return `valid`.

In Section 4 and Section 5 we give two black box tracing algorithms. The first only requires minimal black box access to the decoder. The second requires full black box access.

Our definition of minimal access fits well with the music player analogy. Consider a music player that takes an encrypted music clip and either plays the music or says that the clip is invalid. The music player must play all well formed music clips. A well formed music clip is formated as $\langle C, E_K(S)\rangle$ where E is a semantically secure encryption scheme, K is a symmetric key used to encrypt the music S, and C is a proper encryption of K using the traitor tracing system. Suppose the traitor tracing scheme supports minimal access black box tracing. We trace a pirate music player as follows: when the tracer issues a query $\langle C, K\rangle$ we pick a random peice of music S and create the music clip $\langle C, E_K(S)\rangle$. If the player plays the music clip we respond to the tracer query with `valid`. Otherwise, we respond with `invalid`. The tracing algorithm will correctly identify one of the pirate keys.

Throughout the paper we assume the pirate boxes are stateless. That is, the pirate decoder does not respond to decryption requests based on its responses to previous requests. This does not seem to be a problem since the scheduling of tracing queries can be randomized, and furthermore, tracer queries can be interspersed within regular decryption requests.

We point out that our definition of the pirate says that the pirate obtains a *random* subset of size k of the ℓ private keys. The pirate does not choose which private keys he receives. This is a a natural definition since we are assuming that keys are assigned to users in a random fashion: when a new user buys a decryption box the user receives a box containing a randomly chosen unassigned private key. Hence, even if the pirate breaks into boxes belonging to k users *of his choice*, the pirate still obtains a set of k random keys.

Representations: Our traitor tracing scheme relies on the *representation problem*. When $y = \prod_{i=1}^{2k} h_i^{\delta_i}$ we say that $(\delta_1, \ldots, \delta_{2k})$ is a "representation" of y with respect to the base $h_1, \ldots, h_{2k}$. If $\bar{d}_1, \ldots, \bar{d}_m$ are representations of y with respect to the same base, then so is any "convex combination" of the representations: $\bar{d} = \sum_{i=1}^{m} \alpha_i \bar{d}_i$ where $\alpha_1, \ldots, \alpha_m$ are scalars such that $\sum_{i=1}^{m} \alpha_i = 1$.

Notation: We let $\bar{u} \cdot \bar{v}$ denote the inner product of the two vectors $\bar{u}$ and $\bar{v}$.

3 The encryption scheme

We are now ready to present our tracing traitor encryption scheme. Let s be a security parameter and k be the maximal coalition size. Our scheme defends against any collusion of at most k parties. We wish to generate one public key and ℓ corresponding private keys. Without loss of generality we assume $\ell \geq 2k + 2$ (if $\ell < 2k + 2$ we set $\ell = 2k + 2$ and generate ℓ private keys).

Our scheme makes use of a certain *linear space tracing* code Γ which is a collection of ℓ codewords in $\mathbb{Z}^{2k}$. The construction of the set Γ and the properties it has to satisfy are described in the next section. For now, it suffices to view the ℓ words in Γ as vectors of integers of length $2k$. The set $\Gamma = \{\gamma^{(1)}, \ldots, \gamma^{(\ell)}\}$ is fixed and publicly known.

Let G_q be a group of prime order q. The security of our encryption scheme relies on the difficulty of computing discrete log in G_q. More precisely, the security is based on the difficulty of the Decision Diffie-Hellman problem [3] in G_q as discussed below. One can take as G_q the subgroup of $\mathbb{Z}_p^*$ of order q where p is a prime with $q|p-1$. Alternatively, one can use the group of points of an elliptic curve over a finite field.

Key generation: Perform the following steps:

1. Let $g \in G_q$ be a generator of G_q.
2. For $i = 1, \ldots, 2k$ choose a random $r_i \in \mathbb{F}_q$ and compute $h_i = g^{r_i}$.
3. The public key is $\langle y, h_1, \ldots, h_{2k} \rangle$, where $y = \prod_{i=1}^{2k} h_i^{\alpha_i}$ for random $\alpha_1, \ldots, \alpha_{2k} \in \mathbb{F}_q$.
4. A private key is an element $\theta_i \in \mathbb{F}_q$ such that $\theta_i \cdot \gamma^{(i)}$ is a representation of y with respect to the base $h_1, \ldots, h_{2k}$. The i'th key, θ_i, is derived from the i'th codeword $\gamma^{(i)} = (\gamma_1, \ldots, \gamma_{2k}) \in \Gamma$ by

$$\theta_i = (\sum_{j=1}^{2k} r_j \alpha_j) / (\sum_{j=1}^{2k} r_j \gamma_j) \pmod{q} \tag{1}$$

To simplify the exposition we frequently refer to the private key as being the representation $\bar{d}_i = \theta_i \cdot \gamma^{(i)}$. Note however that only θ_i needs to be kept secret since the code Γ is public. One can verify that $\bar{d}_i$ is indeed a representation of y with respect to the base $h_1, \ldots, h_{2k}$.[1]

Encryption: To encrypt a message M in G_q do the following: first pick a random element $a \in \mathbb{F}_q$. Set the ciphertext C to be

$$C = \langle M \cdot y^a, \ h_1^a, \ \ldots, \ h_{2k}^a \rangle$$

Decryption: To decrypt a ciphertext $C = \langle S, H_1, \ldots, H_{2k} \rangle$ using user i'th secret key, θ_i, compute

$$M = S / U^{\theta_i} \qquad \text{where} \qquad U = \prod_{j=1}^{2k} H_j^{\gamma_j}$$

Here $\gamma^{(i)} = (\gamma_1, \ldots, \gamma_{2k}) \in \Gamma$ is the codeword from which θ_i is derived. The cost of computing U is far less than $2k + 1$ exponentiations thanks to simultaneous multiple exponentiation [10, p. 618]. Also note that U can be computed without knowledge of the private key, leaving only a single exponentiation by the private key holder to complete the decryption.

Before going any further we briefly show that the encryption scheme is sound, i.e. any private key θ_i correctly decrypts any ciphertext. Given a ciphertext $C = \langle M \cdot y^a, h_1^a, \ldots, h_{2k}^a \rangle$, decryption will yield $M \cdot y^a / U^{\theta_i}$ where $U = \prod_{j=1}^{2k} (h_j^a)^{\gamma_j}$. Then

$$U^{\theta_i} = (\prod_{j=1}^{2k} g^{a r_j \gamma_j})^{\theta_i} = (g^{\Sigma_{j=1}^{2k} r_j \gamma_j})^{\theta_i a} = (g^{\Sigma_{j=1}^{2k} r_j \alpha_j})^a = (\prod_{j=1}^{2k} h_j^{\alpha_j})^a = y^a$$

as needed. The third equality follows from Equation (1). More generally, it is possible to decrypt given any representation $(\delta_1, \ldots, \delta_{2k})$ of y with respect to the base $h_1, \ldots, h_{2k}$, since $\prod_{j=1}^{2k} (h_j^a)^{\delta_j} = y^a$.

Tracing algorithm: We describe our tracing algorithm in Section 3.3.

[1] A codeword might not have an associated private key in the extremely unlikely event that the denominator is zero in the calculation of θ_i.

3.1 Proof of security

We now show that our encryption scheme is semantically secure against a passive adversary assuming the difficulty of the Decision Diffie-Hellman problem (DDH) in G_q. The assumption says that in G_q, no polynomial time statistical test can distinguish with non negligible advantage between the two distributions $D = \langle g_1, g_2, g_1^a, g_2^a \rangle$ and $R = \langle g_1, g_2, g_1^a, g_2^b \rangle$ where g_1, g_2 are chosen at random in G_q and a, b are chosen at random in $\mathbb{F}_q$.

Theorem 3.1 *The encryption scheme is semantically secure against a passive adversary assuming the difficulty of DDH in G_q.*

Proof Suppose the scheme is not semantically secure against a passive adversary. Then there exists an adversary that given the public key $\langle y, h_1, \ldots, h_{2k} \rangle$ produces two messages $M_0, M_1 \in G_q$. Given the encryption C of one of these messages the adversary can tell with non-negligible advantage ϵ which of the two messages he was given. We show that such an adversary can be used to decide DDH in G_q. Given $\langle g_1, g_2, u_1, u_2 \rangle$ we perform the following steps to determine if it is chosen from R or D:

Step 1: Choose random $r_2, \ldots, r_{2k} \in \mathbb{F}_q$. Set $y = g_1$, $h_1 = g_2$, and $h_i = g_2^{r_i}$ for $i = 2, \ldots, 2k$.

Step 2: Give $\langle y, h_1, \ldots, h_{2k} \rangle$ to the adversary. Adversary returns $M_0, M_1 \in G_q$.

Step 3: Pick a random $b \in \{0, 1\}$ and construct the ciphertext

$$C = \langle M_b u_1, u_2, u_2^{r_2}, \ldots, u_2^{r_{2k}} \rangle$$

Step 4: Give the ciphertext C to the adversary. Adversary returns $b' \in \{0, 1\}$.

Step 5: If $b = b'$ output "D". Otherwise output "R".

Observe that if the tuple $\langle g_1, g_2, u_1, u_2 \rangle$ is chosen from D, then the ciphertext C is an encryption of M_b. If the quadruple is from R, then the ciphertext is an encryption of $M_b g_1^{(a_1 - a_2)}$, where $u_1 = g_1^{a_1}$ and $u_2 = g_2^{a_2}$. In other words, the ciphertext is the encryption of a random message. Hence $b = b'$ holds with probability $1/2$. By a standard argument, a non-negligible success probability for the adversary implies a non-negligible success probability in deciding DDH. □

3.2 Constructing new representations

To decrypt, it suffices to know any representation of y with respect to the base $h_1, \ldots, h_{2k}$. We have already noted that if $\bar{d}_1, \ldots, \bar{d}_m \in \mathbb{F}_q^{2k}$ are representations of y then any convex combination of $\bar{d}_1, \ldots, \bar{d}_m$ is also a representation of y. The following lemma shows that convex combinations are the only new representations of y that can be efficiently constructed from $\bar{d}_1, \ldots, \bar{d}_m \in \mathbb{F}_q^{2k}$.

Lemma 3.2 *Let $\langle y, h_1, \ldots, h_{2k} \rangle$ be a public key. Suppose an adversary is given the public key and m private keys $\bar{d}_1, \ldots, \bar{d}_m \in \mathbb{F}_q^{2k}$ for $m < 2k - 1$. If the adversary can generate a new representation $\bar{d}$ of y with respect to the base $h_1, \ldots, h_{2k}$ that is not a convex combination of $\bar{d}_1, \ldots, \bar{d}_m$ then the adversary can compute discrete logs in G_q.*

Proof Let g be a generator of G_q. Suppose we are given $z = g^x$. We show how to use the adversary to compute x. Choose random $b, r_1, \ldots, r_{2k}, s_1, \ldots, s_{2k} \in Z_q$. Construct the set $\{h_1, \ldots, h_{2k}\}$ where $h_i = z^{r_i} g^{s_i}$ for all i, $1 \leq i \leq 2k$. Compute $y = g^b$. Find m linearly independent (and otherwise random) solutions $\bar{\alpha}_1, \ldots, \bar{\alpha}_m$ to $\bar{\alpha} \cdot \bar{r} = 0 \bmod q$ and $\bar{\alpha} \cdot \bar{s} = b \bmod q$. These m vectors are representations

of y with respect to the base $h_1, \ldots, h_{2k}$. Suppose that the adversary can find another representation $\bar{\beta}$ that is not a convex combination of $\bar{\alpha}_1, \ldots, \bar{\alpha}_m$. Then $\bar{\beta} \cdot \bar{r} \neq 0 \bmod q$ with probability at least $1 - \frac{1}{q}$. But then $(b - \bar{\beta} \cdot \bar{s})(\bar{\beta} \cdot \bar{r})^{-1}$ is the discrete log of z. □

3.3 Non-Black-Box Tracing

We now turn our attention to a tracing algorithm for our basic encryption scheme. Throughout this subsection we assume the pirate decoder contains at least one representation of y. Furthermore, we assume that by examining the decoder implementation it is possible to obtain one of these representations, $\bar{d}$. Hence, the tracing algorithm presented in this section is not black box: it assumes one can extract at least one representation of y from the pirate decoder. Whenever it is easy to reverse engineer the pirate decoder this algorithm directly exposes the keys used by the pirate. In the next section we show how to convert this algorithm into a black-box tracing algorithm that exposes the pirate keys without reverse engineering the pirate decoder.

Suppose the pirate obtains k keys $\bar{d}_1, \ldots, \bar{d}_k$. Let $\bar{d}$ be the representation of y found in the pirate decoder. Then by Lemma 3.2 we know that $\bar{d}$ must lie in the linear span of the representations $\bar{d}_1, \ldots, \bar{d}_k$. We construct a tracing algorithm that given $\bar{d}$ outputs one of $\bar{d}_1, \ldots, \bar{d}_k$.

Recall that the construction of private keys made use of a set $\Gamma \subseteq \mathbb{F}_q^{2k}$ containing ℓ codewords. Each of the ℓ users is given a private key $\bar{d}_i \in \mathbb{F}_q^{2k}$ which is a multiple of a codeword in Γ. To solve the tracing problem we must construct a set $\Gamma \subseteq \mathbb{F}_q^{2k}$ containing ℓ codewords with the following property. Let $\bar{d}$ be a point in the linear span of some k codewords $\gamma^{(1)}, \ldots, \gamma^{(k)} \in \Gamma$. Then at least one γ in $\gamma^{(1)}, \ldots, \gamma^{(k)}$ must be a member of any coalition (of at most k users) that can create $\bar{d}$. This γ identifies one of the private keys that *must* have participated in the construction of the pirated key $\bar{d}$. Furthermore, there should exist an efficient tracing algorithm that when given $\bar{d}$ as input, outputs γ. In fact, our tracing algorithm will output every $\gamma^{(i)}$ that has nonzero weight in the linear combination.

The set Γ: We begin by describing the set Γ containing ℓ codewords over $\mathbb{F}_q^{2k}$. Since q is a large prime we may assume $q > \max(\ell, 2k)$. Consider the following $(\ell - 2k) \times \ell$ matrix:

$$A = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 2 & 3 & \ldots & \ell \\ 1^2 & 2^2 & 3^2 & \ldots & \ell^2 \\ 1^3 & 2^3 & 3^3 & \ldots & \ell^3 \\ & \vdots & & & \vdots \\ 1^{\ell-2k-1} & 2^{\ell-2k-1} & 3^{\ell-2k-1} & \ldots & \ell^{\ell-2k-1} \end{pmatrix} \pmod q$$

Observe that any vector in the span of the rows of A corresponds to a polynomial of degree at most $\ell - 2k - 1$ evaluated at the points $1, \ldots, \ell$.

Let $b_1, \ldots, b_{2k}$ be a basis of the linear space of vectors satisfying $A\bar{x} = 0 \bmod q$. Viewing these $2k$ vectors as the columns of a matrix we obtain an $\ell \times 2k$ matrix B:

$$B = \begin{pmatrix} | & | & | & & | \\ b_1 & b_2 & b_3 & \ldots & b_{2k} \\ | & | & | & & | \end{pmatrix}$$

We define Γ as the set of rows of the matrix B. Hence, Γ contains ℓ codewords each of length $2k$. We note that using Lagrange interpolation one can directly construct the i'th codeword in Γ using approximately ℓ arithmetic operations modulo q.

Non-Black-Box tracing algorithm: Consider the set of vectors in Γ. Let $\bar{d} \in \mathbb{F}_q^{2k}$ be a vector formed by taking a linear combination of at most k vectors in Γ. We show that given $\bar{d}$ one can efficiently determine the unique set of vectors in Γ used to construct $\bar{d}$. Since the vectors in Γ form the rows of the matrix B above we know there exists a vector $\bar{w} \in \mathbb{F}_q^{\ell}$ of Hamming weight at most k such that $\bar{w} \cdot B = \bar{d}$. We show how to recover the vector $\bar{w}$ given $\bar{d}$.

Step 1: Find a vector $\bar{v} \in \mathbb{F}_q^{\ell}$ such that $\bar{v} \cdot B = \bar{d}$. Many such vectors exist. Choose one arbitrarily.[2] Since $(\bar{v} - \bar{w}) \cdot B = 0$ we know that $\bar{v} - \bar{w}$ is in the linear span of the rows of the matrix A (the rows of A span the space of vectors orthogonal to the columns of B). In other words, there exists a unique polynomial $f \in \mathbb{F}_q[x]$ of degree at most $\ell - 2k - 1$ such that $\bar{v} - \bar{w} = \langle f(1), \ldots, f(\ell) \rangle$.

Step 2: Since $\bar{w}$ has Hamming weight at most k, we know that $\langle f(1), \ldots, f(\ell) \rangle$ equals $\bar{v}$ in all but k components. Hence, using Berlekamp's algorithm [2] we can find f from $\bar{v}$. The polynomial f gives us the vector $\bar{v} - \bar{w}$ from which we recover $\bar{w}$ as required.

For completeness we briefly recall Berlekamp's algorithm. The algorithm enables us to find f given the vector $\bar{v} \in \mathbb{F}_q^{\ell}$. Let g be a polynomial of degree at most k such that $g(i) = 0$ for all $i = 1, \ldots, \ell$ for which $f(i) \neq v_i$ (where v_i is the i'th component of $\bar{v}$). Then we know that for all $i = 1, \ldots, \ell$ we have $f(i)g(i) = g(i)v_i$. The polynomial fg has degree at most $\ell - k - 1$. Hence, we get ℓ equations (for each of $i = 1, \ldots, \ell$) in ℓ variables (the variables are the coefficients of the polynomials fg and g, where the leading coefficient of g is 1). Let h and g be a solution where g is a non-zero polynomial: h is a polynomial of degree at most $\ell - k - 1$ and g is of degree at most k. We know that whenever $f(i) = v_i$ (i.e. at $\ell - k$ points) we have $h(i) = g(i)v_i = g(i)f(i)$. It follows that $f = h/g$.

This completes the description of the tracing algorithm. Our tracing algorithm satisfies several properties:

Error free tracing The tracing algorithm is deterministic in the sense that there is no error probability. Any key output by the tracing algorithm must have participated in the construction of the pirated key.

Beyond threshold tracing If more than k parties colluded to create the pirated key, then Berlekamp's algorithm may fail to recover the polynomial f, and tracing will fail. Above this bound, recent results of Guruswami and Sudan [8] may be used to output a list of candidate polynomials for f. The tracer gets a list of "leads" for the fraud investigation that includes the actual colluders. This will be effective against coalitions of size at most $2k - 1$.

Running time The tracing algorithm requires that we solve a linear system of dimension ℓ (the total number of users). A naive implementation runs in time $O(\ell^2)$ (field operations). Asymptotically efficient versions of Berlekamp's algorithm run in time $\tilde{O}(\ell)$, where the "soft-Oh" notation hides polylog terms. The fastest known algorithm, due to Pan [13], runs in time $O(\ell \log \ell \log\log \ell)$.

Our scheme also supports two types of black box tracing techniques, which we describe in the next two sections.

[2] If B is in canonical form with the identity matrix as its first $2k$ rows, then $\bar{v} = (\bar{d} || 0 \ldots 0)$ suffices.

4 Minimal Access Black box tracing against arbitrary pirates

In this section, we show a minimal access black box tracing algorithm for our basic encryption scheme that works against arbitrary pirates. To achieve this we introduce an easier tracing goal called *black box confirmation.*

4.1 Black Box Confirmation

When the police finds a pirate decoder they often have a suspect pirate in mind. The goal of black box confirmation is to allow the police to confirm their suspicion. A black box confirmation algorithm is defined as follows: the algorithm is given (1) the public key and all random bits used during initial key generation, (2) black box access to a pirate decoder $\mathcal{D}$, and (3) a set $T_{suspect}$ of private keys suspected of creating $\mathcal{D}$. The black box confirmation algorithm outputs either "`not-guilty`" or "`key` $\bar{d}$ `is guilty`" for some $\bar{d} \in T_{suspect}$. Let $T_{\mathcal{D}}$ be the set of private keys in the pirate's possession when $\mathcal{D}$ was created. The output must satisfy the following two requirements:

1. **Confirmation:** If $T_{\mathcal{D}} \subseteq T_{suspect}$ then the confirmation algorithm must pronounce at least one key $\bar{d} \in T_{suspect}$ as guilty.
2. **Soundness:** If the confirmation algorithm outputs `key` $\bar{d}$ `is guilty` then $\bar{d} \in T_{\mathcal{D}}$.

As always we assume that the size of both $T_{suspect}$ and $T_{\mathcal{D}}$ is less than the collusion bound k. The police does not know ahead of time whether $T_{\mathcal{D}} \subseteq T_{suspect}$. Hence, condition (2) ensures that when $T_{\mathcal{D}} \not\subseteq T_{suspect}$ the pirate cannot fool the algorithm into accusing an innocent user. Note that when the police has a small number of suspect sets $T^1_{suspect}, \ldots, T^r_{suspect}$ they can easily find a guilty key $\bar{d} \in T_{\mathcal{D}}$ using black box confirmation.

A black box confirmation algorithm will imply a black box tracing algorithm as follows. Run the confirmation algorithm on all $\binom{\ell}{k}$ candidate coalitions (ℓ is the total number of users in the system). By the confirmation property (Property 1) when the tested coalition contains the guilty coalition, some member of the guilty coalition will be pronounced guilty. A guilty user might be caught when other coaltions are tested. By the soundess property (property 2) whenever a key $\bar{d}$ is pronouned guilty the key is a member of the guilty coaltion with very high probability. Thus we obtain a black box tracing algorithm with a running time of $O(\binom{\ell}{k}k^2)$. This is not an efficient tracing algorithm, but it shows that black box tracing is possible in principle. Hence, we do not have to rely on the pirate to embed specific information in the pirate decocer. In the next section we show an efficient black box tracing algorithm against a restricted set of pirates.

We briefly outline the basic idea behind our black box confirmation algorithm. Let $T_{suspect} = \{\bar{d}_1, \ldots, \bar{d}_k\}$ and let g be a generator of G_q. Let $\langle y, h_1, \ldots, h_{2k}\rangle$ be a public key. To confirm its suspicion of $T_{suspect}$ the tracer queries the pirate decoder with a random invalid ciphertext $\tilde{C} = \langle S, g^{z_1}, \ldots, g^{z_{2k}}\rangle$, where the vector $\bar{z}$ satisfies $\bar{z} \cdot \bar{d}_i = w$ for all $i \in T_{suspect}$. Here w is a random element of $\mathbb{F}_q$. This ciphertext is invalid since $\langle g^{z_1}, \ldots, g^{z_{2k}}\rangle$ is most likely not of the form $\langle h_1^r, \ldots, h_{2k}^r\rangle$ for any r. We show that when $T_{\mathcal{D}} \subseteq T_{suspect}$ the pirate decoder cannot distinguish this invalid ciphertext from a real one. Consequently, we show that it must respond with $A = S/g^{\bar{z}\cdot\bar{d}}$ where $\bar{d}$ is some representation of y in the convex hull of $\bar{d}_1, \ldots, \bar{d}_k$. Hence, when $T_{\mathcal{D}} \subseteq T_{suspect}$ the pirate decoder always responds with $A = S/g^w$. When $T_{\mathcal{D}} \cap T_{suspect} = \emptyset$ the decoder's view is independent of S/g^w. Hence, the tracer can tell if $T_{suspect}$ contains one of the pirate's keys. With a bit more work the tracer can then point to a specific $\bar{d} \in T_{suspect}$ that must be in the pirate's possession.

4.2 Definitions and Distributions

Before we present the black box confirmation algorithm we need to define a few terms. For a given suspect set $T_{suspect} = \{\bar{d}_1, \ldots, \bar{d}_k\} \subseteq \mathbb{F}_q^{2k}$ define the following terms:

- For $i = 0, \ldots, k$ define $T_i = \{\bar{d}_1, \ldots, \bar{d}_i\} \subseteq T_{suspect}$. T_0 is the empty set.
- For $i = 0, \ldots, k$, and $W \in G_q$ define $\mathsf{CT}_i(W)$ as the set:

$$\mathsf{CT}_i(W) = \left\{ C = \langle S, H_1, \ldots, H_{2k} \rangle \;\middle|\; S \in G_q \text{ and } W = \prod_{i=1}^{2k} H_i^{\delta_i} \text{ for all } \bar{d} = \langle \delta_1, \ldots, \delta_{2k} \rangle \in T_i \right\}$$

  Define $\mathsf{CT}_i = \bigcup_{W \in G_q} \mathsf{CT}_i(W)$.
  Note that when key $\bar{d} \in T_i$ is used to decrypt $C \in \mathsf{CT}_i(W)$ we get S/W.
- For $i = 0, \ldots, k$ define the distribution CW_i on pairs $\langle W, C \rangle$ as follows: pick a random $W \in G_q$, pick a random $C \in \mathsf{CT}_i(W)$, output $\langle W, C \rangle$.
- For $i = 0, \ldots, k$ define $P_{\mathcal{D},i}$ as: $P_{\mathcal{D},i} = \Pr_{\langle W,C \rangle \in \mathsf{CW}_i}[\mathcal{D}(C, S/W) = \texttt{valid}]$.
  Where $C = \langle S, H_1, \ldots, H_{2k} \rangle$. Here $\langle W, C \rangle$ is chosen from the distribution CW_i and $\mathcal{D}(C, S/W)$ denotes the success or failure of the pirate decoder on input $C, S/W$ (minimal access).
- Given a public key $\langle y, h_1, \ldots, h_{2k} \rangle$ we refer to the set $\{\langle S, h_1^r, \ldots, h_{2k}^r \rangle : r \in \mathbb{F}_q\}$ as the set of valid ciphertexts. We refer to ciphertexts outside this set as invalid ciphertexts.
  Define the distribution CW_{valid} on pairs $\langle W, C \rangle$ as follows: (1) pick a random $r \in \mathbb{F}_q$ and $S \in G_q$, (2) set $W = y^r$, and $C = \langle S, h_1^r, \ldots, h_{2k}^r \rangle$, (3) output $\langle W, C \rangle$. Note that C is a valid ciphertext and its decryption is S/W.

Note that for any i we can efficiently sample from the distribution CW_i as follows: (1) pick a random $w \in \mathbb{F}_q$ and set $W = g^w$, (2) pick a random vector $\bar{z} = \langle z_1, \ldots, z_{2k} \rangle \in \mathbb{F}_q^{2k}$ such that $\bar{z} \cdot \bar{d} = w$ for all $\bar{d} \in T_i$, (3) pick a random $S \in G_q$, (4) set $C = \langle S, g^{z_1}, \ldots, g^{z_{2k}} \rangle$, and output $\langle W, C \rangle$.

We claim that $P_{\mathcal{D},0} \leq 1/q$. To see this observe that when $\langle W, C \rangle$ is chosen from CW_0 we have that W is independent of C. It follows that $\mathcal{D}(C, S/W)$ is valid with probability at most $1/q$.

4.3 Description of Black Box Confirmation Algorithm

We are now ready to describe the black box confirmation algorithm. An outline of the algorithm is given in [4]. The algorithm is given minimal access to $\mathcal{D}$ and works as follows:

Algorithm 1:

Step 1: Compute estimates for every $P_{\mathcal{D},i}$.
For each $i = 0, \ldots, k$ do:

Step 1a: For $j = 1, \ldots, \lambda$ pick random and independent pairs $\langle W_j, C_j \rangle$ chosen according to the distribution CW_i. The value of λ will be determined later.

Step 1b: Run $\mathcal{D}$ on all inputs $C_1, \ldots, C_\lambda$ and let c_i be the number of times that $\mathcal{D}(C_j, S_j/W_j) =$ `valid`.

Step 1c: Set $\hat{p}_i = c_i/\lambda$.

Step 2: If $\hat{p}_k \leq 3/4$ output `not-guilty` and stop.

Step 3: Otherwise, since $P_{\mathcal{D},0} \leq 1/q$ it follows that $\hat{p}_0 \leq 1/4$ with high probability (we calculate exact probabilities below). Then:

$$\frac{1}{2} \leq |\hat{p}_k - \hat{p}_0| = \left| \sum_{i=1}^{k} \hat{p}_i - \hat{p}_{i-1} \right| \leq \sum_{i=1}^{k} |\hat{p}_i - \hat{p}_{i-1}|$$

It follows that there exists a $1 \leq j \leq k$ such that: $|\hat{p}_j - \hat{p}_{j-1}| \geq \frac{1}{2k}$.
Output $\bar{d}_j \in T_{suspect}$ is guilty, and stop.

The completes the description of the black box confirmation algorithm. The algorithm requires $\lambda(k+1)$ queries to the pirate decoder $\mathcal{D}$. As we show in Lemma 4.6, we take λ on the order of $k \log k$ and hence the algorithm has running time $O(k^2 \log k)$. Note that Algorithm 1 does not need the public key or the private bits used during key generation, only the k private keys of the suspect coalition.

In the rest of this section, we will prove the following theorem.

Theorem 4.1 *Algorithm 1 is a black box confirmation algorithm.*

To prove Theorem 4.1, we must show that Algorithm 1 satisfies the confirmation property (Property 1) and the soundness property (Property 2) for black box confirmation. These will be proven in Lemma 1 and Lemma 4.6 respectively. The probability space in both lemmas is over the random bits used by Algorithm 1, the random bits used by the pirate and pirate decoder, and the random set of k private keys given to the pirate.

4.4 Algorithm 1 Satisfies the Confirmation Property

We show that Algorithm 1 satisfies the confirmation property, i.e., no pirate can fool Algorithm 1 when $T_{\mathcal{D}} \subseteq T_{suspect}$. To do so, we show that the following *non-traceable pirate* does not exist.

Non-traceable pirate: Let $\mathcal{P}$ be some pirate. Let $\langle y, h_1, \ldots, h_{2k} \rangle$ be a public key and let $T_{\mathcal{D}} = \{\bar{d}_1, \ldots, \bar{d}_k\}$ be a random set of k private keys. We say that the pirate $\mathcal{P}$ is *ϵ-non-traceable* if given the public key and $T_{\mathcal{D}}$, the pirate creates a pirate decoder $\mathcal{D}$ with the following properties: (1) $\mathcal{D}$ correctly decrypts all valid ciphertexts, and (2) when $T \subseteq T_{suspect}$ the pirate decoder $\mathcal{D}$ causes Algorithm 1 to output "`not-guilty`" with proability at least ϵ. The following lemma shows that a non-traceable pirate does not exist. Recall that u is the number of samples used in Step 1 of Algorithm 1.

Lemma 4.2 *Let $\epsilon > 0$ and let $\lambda = 1$. Suppose $\mathcal{P}$ is an ϵ-non-traceable pirate. Then $\mathcal{P}$ can be used to distinshuish DH tuples from random tuples with advantage ϵ.*

Proof Suppose $T_{\mathcal{D}} \subseteq T_{suspect}$. Then, by assumption, when Algorithm 1 is given a decoder $\mathcal{D}$ created by $\mathcal{P}$, Algorithm 1 outputs `not-guilty` with probability at least ϵ. This means $\hat{p}_k \leq 3/4$ with probability at least ϵ. Since $\lambda = 1$ it follows that $P_{\mathcal{D},k} < 1 - \epsilon$. This means that the pirate $\mathcal{P}$ is able to build a pirate decoder $\mathcal{D}$ that correctly decrypts all valid ciphertexts; however for a random $C \in \mathsf{CT}_k(W)$ it outputs "invalid" on input $C, S/W$ with probability at least ϵ.

We show how the pirate $\mathcal{P}$ can be used to solve DDH in G_q. Given a challenge tuple $\langle g_1, g_2, u_1, u_2 \rangle$ we use the following algorithm to decide whether it is a random tuple or a Diffie-Hellman tuple. We write $g_2 = g_1^c$ for some unknown c.

Algorithm $\mathcal{A}$:

Step 1: Choose a random set of k codewords in Γ. Denote these by $\gamma^{(1)}, \ldots, \gamma^{(k)} \in \mathbb{F}_q^{2k}$. (These will correspond to the k private keys given to the pirate.)

Step 2: Pick a random $A \in \mathbb{F}_q$ such that $g_1^A g_2 \neq 1$.

Step 3: Pick random vectors $\bar{a}, \bar{b} \in \mathbb{F}_q^{2k}$ such that the vector $\bar{a} - A\bar{b}$ is orthogonal to the k vectors $\gamma^{(1)}, \ldots, \gamma^{(k)}$. Write $\bar{a} = (a_1, \ldots, a_{2k})$ and $\bar{b} = (b_1, \ldots, b_{2k})$.

Step 4: Set $h_i = g_1^{a_i} g_2^{b_i}$ for $i = 1, \ldots, 2k$.

Step 5: Pick a random vector $\bar{\alpha} \in \mathbb{F}_q^{2k}$ such that $\bar{\alpha}$ is orthogonal to $\bar{a} - A\bar{b}$. Write $\bar{\alpha} = (\alpha_1, \ldots, \alpha_{2k})$.

Step 6: Set $y = h_1^{\alpha_1} \cdots h_{2k}^{\alpha_{2k}}$. Note that $y = g_1^{\bar{\alpha}\cdot\bar{a}} g_2^{\bar{\alpha}\cdot\bar{b}} = g_1^{A(\bar{\alpha}\cdot\bar{b})} g_2^{\bar{\alpha}\cdot\bar{b}} = (g_1^A g_2)^{\bar{\alpha}\cdot\bar{b}}$.

Step 7: For $i = 1, \ldots, k$ set $\theta_i = \dfrac{\bar{\alpha} \cdot \bar{b}}{\gamma^{(i)} \cdot \bar{b}}$. Set $\bar{d}_i = \theta_i \gamma^{(i)}$.

Observe that $\bar{d}_1, \ldots, \bar{d}_k$ are representations of y to the base $h_1, \ldots, h_{2k}$. Indeed, for $\bar{d}_i = (\delta_1, \ldots, \delta_{2k})$ we have:

$$\prod_{i=1}^{2k} h_i^{\delta_i} = g_1^{\bar{a}\cdot\bar{d}_i} g_2^{\bar{b}\cdot\bar{d}_i} = \left[g_1^{\bar{a}\cdot\gamma^{(i)}} g_2^{\bar{b}\cdot\gamma^{(i)}}\right]^{\theta_i} = \left[g_1^{A(\bar{b}\cdot\gamma^{(i)})} g_2^{(\bar{b}\cdot\gamma^{(i)})}\right]^{\theta_i} = \left(g_1^A g_2\right)^{\bar{b}\cdot\bar{a}} = y$$

Step 8: Run the pirate $\mathcal{P}$ giving it the public key $\langle y, h_1, \ldots, h_{2k}\rangle$ and the k private keys $\theta_1, \ldots, \theta_k$. The pirate builds a pirate decoder $\mathcal{D}$.

Step 9: Pick a random $S \in G_{\tilde{q}}$ and construct the ciphertext: $\tilde{C} = \langle S,\ {u_1}^{a_1} {u_2}^{b_1},\ \ldots,\ {u_1}^{a_{2k}} {u_2}^{b_{2k}}\rangle$
Let $\tilde{C} = \langle S, H_1, \ldots, H_{2k}\rangle$ and set $W = \prod H_i^{\alpha_i}$

Step 10: Query the pirate decoder $\mathcal{D}$ on the ciphertext/message pair $\langle \tilde{C},\ S/W\rangle$. If $\mathcal{D}$ returns "valid" output that $\langle g_1, g_2, u_1, u_2\rangle$ is a Diffie-Hellman tuple. Otherwise, output that $\langle g_1, g_2, u_1, u_2\rangle$ is a random tuple.

This concludes the description of algorithm $\mathcal{A}$ for solving DDH. We show that algorithm $\mathcal{A}$ above correctly decides whether the given tuple $\langle g_1, g_2, u_1, u_2\rangle$ is a DDH tuple. This follows from the following three claims.

Claim 4.3 *The distribution on the public key $\langle y, h_1, \ldots, h_{2k}\rangle$ and the k private keys $\theta_1, \ldots, \theta_k$ given to the pirate $\mathcal{P}$ in Step 8 is identical to the distribution generated by the real key generation algorithm.*

Proof Consider the key generation algorithm of Section 3. Observe that given $h_1, \ldots, h_{2k}$ and y the values $\theta_1, \ldots, \theta_k$ are determined. Indeed, given the values $y, h_1, \ldots, h_{2k}$ the vector $r_1, \ldots, r_{2k} \in \mathbb{F}_q$ and $\sum_{i=1}^{2k} r_i \alpha_i$ are determined. Then the θ_i are determined from equation (1). Hence, when using the real key generation algorithm the probability of seeing a specific valid sequence $\langle y, h_1, \ldots, h_{2k}, \theta_1, \ldots, \theta_k\rangle$ is:

$$\Pr[\langle y, h_1, \ldots, h_{2k}, \theta_1, \ldots, \theta_k\rangle] = 1/q^{2k+1}$$

where the probability is over the random bits used by the key generation algorithm.

We show that the same holds for the public key given to the pirate in Step 8. Let Γ_0 be the $k \times 2k$ matrix whose rows are the vectors $\gamma^{(1)}, \ldots, \gamma^{(k)} \in \mathbb{F}_q^{2k}$. This matrix has rank k. We first show that given the k linear constraints on $\bar{a}, \bar{b}$ in Step 3, the vector $\langle h_1, \ldots h_{2k}\rangle$ is uniformly distributed in G_q^{2k}. Since $h_i = g_1^{a_i} g_2^{b_i} = g_1^{a_i + cb_i}$ it suffices to argue that given the k constraints in Step 3 the $2k$ values $a_i + cb_i$ are uniformly and independently distributed in F_q^{2k}. To do so it suffices to prove that the set of $3k$ relations $(\bar{a} - A\bar{b}) \cdot \gamma^{(i)} = 0$ for $i = 1, \ldots, k$ and $a_j + cb_j = 0$ for $j = 1, \ldots, 2k$ are linearly independent. In other words, we need to show that the rows of the following $3k \times 4k$ matrix are

linearly independent:

$$\Omega = \begin{pmatrix} \Gamma_0 & -A\Gamma_0 \\ I_{2k} & cI_{2k} \end{pmatrix}$$

Here I_{2k} denotes the $2k \times 2k$ unit matrix. One can immediately verify that whenever $c \neq -A$ the matrix Ω has rank $3k$ as required. We know that $c \neq -A$ since in Step 2 we ensure that $g_1^A g_2 \neq 1$. This shows that the vector $\langle h_1, \ldots, h_{2k} \rangle$ is uniformly distributed in G_q^{2k}.

Next, we show that y is uniformly distributed in G_q and is independent of $\langle h_1, \ldots, h_{2k} \rangle$. This is immediate since $y = (g_1^A g_2)^{(\bar{\alpha} \cdot \bar{b})}$ and $\bar{\alpha} \cdot \bar{b}$ is uniform in $\mathbb{F}_q$. To see that $\bar{\alpha} \cdot \bar{b}$ is uniform (over the choice of $\bar{\alpha}$) recall that in Step 5 $\bar{\alpha}$ is generated as a random vector in $\mathbb{F}_q^{2k}$ subject to one constraint $\bar{\alpha} \cdot (\bar{a} - A\bar{b}) = 0$. This constraint on $\bar{\alpha}$ is linearly independent of the constraint $\bar{\alpha} \cdot \bar{b} = 0$. Hence, since $\bar{\alpha} \cdot \bar{b}$ is independent of $\langle h_1, \ldots, h_{2k} \rangle$, so is y.

Finally, we show that given $y, h_1, \ldots, h_{2k}$ the value of $\theta_1, \ldots, \theta_k$ is determined. Hence, revealing $\theta_1, \ldots, \theta_k$ does not further constrain $\bar{a}, \bar{b}, \bar{\alpha}$. To see this observe that since $\theta_i \gamma^{(i)}$ is a representation of y to the base $h_1, \ldots, h_{2k}$ we have that:

$$\theta_i = \frac{(\bar{a} + c\bar{b}) \cdot \bar{\alpha}}{(\bar{a} + c\bar{b}) \cdot \gamma^{(i)}} = \frac{(A + c) \cdot (\bar{b} \cdot \bar{\alpha})}{(\bar{a} + c\bar{b}) \cdot \gamma^{(i)}}$$

But $(A + c)(\bar{b} \cdot \bar{\alpha})$ is completely determined by y, and the vector $\bar{a} + c\bar{b}$ is completely determined by $h_1, \ldots, h_{2k}$. Thus, given $y, h_1, \ldots, h_{2k}$ the value of $\theta_1, \ldots, \theta_k$ is fixed.

To summarize, the probability that in Step 8 the pirate is given the input $\langle y, h_1, \ldots, h_{2k}, \theta_1, \ldots, \theta_k \rangle$ is $1/q^{2k+1}$ as required. Overall, we have shown that once $\langle y, h_1, \ldots, h_{2k}, \theta_1, \ldots, \theta_k \rangle$ is revealed there are $3k$ relations induced on $\bar{a}, \bar{b}$ and two relations induced on $\bar{\alpha}$ (one by y the other by Step 5). As a result, $\langle \bar{a}, \bar{b} \rangle$ is random in a linear space of dimension k and $\bar{\alpha}$ is random in a linear space of dimension $2k - 2$. This will be used in the next two claims. □

Claim 4.4 *Suppose $\langle g_1, g_2, u_1, u_2 \rangle$ is a random Diffie-Hellman tuple. Then given the public key $\langle y, h_1, \ldots, h_{2k} \rangle$ and the k private keys $\theta_1, \ldots, \theta_k$ the pair $\langle W, \tilde{C} \rangle$ generated in Step 9 is distributed according to CW_{valid}.*

Proof When $\langle g_1, g_2, u_1, u_2 \rangle$ is a Diffie-Hellman tuple we have that $u_1 = g_1^x$ and $u_2 = g_2^x$ for some random unknown x. It follows that $\tilde{C}$ generated in Step 9 satisfies:

$$\tilde{C} = \langle S, (g_1^{a_1} g_2^{b_1})^x, \ldots, (g_1^{a_{2k}} g_2^{b_{2k}})^x \rangle = \langle S, h_1^x, \ldots, h_{2k}^x \rangle$$

Since x and S are random and independent of $\langle y, h_1, \ldots, h_{2k} \rangle$ this is a random valid ciphertext. Furthermore $W = u_1^{\bar{a} \cdot \bar{\alpha}} u_2^{\bar{b} \cdot \bar{\alpha}} = (g_1^A g_2)^{x(\bar{b} \cdot \bar{\alpha})} = y^x$ as required. □

Claim 4.5 *Suppose $\langle g_1, g_2, u_1, u_2 \rangle$ is a random tuple. Then given the public key $\langle y, h_1, \ldots, h_{2k} \rangle$ and the k private keys $\theta_1, \ldots, \theta_k$ the pair $\langle W, \tilde{C} \rangle$ generated in Step 9 is distributed according to CW_k.*

Proof We write $g_2 = g_1^c$ for some unknown c. We also let $u_1 = g_1^{x_1}$ and $u_2 = g_2^{x_2}$, and assume that $x_1 \neq x_2$. We need to show that the pair $\langle W, \tilde{C} \rangle$ generated in Step 9 satisfies: (1) W is uniform in G_q and independent of $I = \langle y, h_1, \ldots, h_{2k}, \theta_1, \ldots, \theta_k \rangle$, and (2) $\tilde{C}$ is uniform in $\mathsf{CT}_k(W)$ and independent of $\langle I, W \rangle$.

We start with W. Define $w = (Ax_1 + cx_2)(\bar{\alpha} \cdot \bar{b})$. Then:

$$W = \prod_{i=1}^{2k} (u_1^{a_i} u_2^{b_i})^{\alpha_i} = u_1^{\bar{a} \cdot \alpha} u_2^{\bar{b} \cdot \alpha} = u_1^{A\bar{b} \cdot \alpha} u_2^{\bar{b} \cdot \alpha} = g_1^{(Ax_1 + cx_2)(\bar{b} \cdot \alpha)} = g^w$$

To see that given I the value W is uniform in G_q it suffices to show that w is uniform in $\mathbb{F}_q$. This is immediate since $\bar{\alpha} \cdot \bar{b} \neq 0$ (since $y \neq 1$) and x_1, x_2 are independent of I.

Next, we show that given I, W the ciphertext $\tilde{C}$ is uniform in $\mathsf{CT}_k(W)$. First, observe that $\tilde{C}$ generated in Step 9 satifies $\tilde{C} \in \mathsf{CT}_k(W)$. To see this write $\tilde{C} = \langle S, H_1, \ldots, H_{2k} \rangle$. Then $H_i = u_1^{a_i} u_2^{b_i}$, and thus for any $\bar{d}_i = \theta_i \gamma^{(i)} = (\delta_1, \ldots, \delta_{2k})$ we have:

$$\prod_{i=1}^{2k} H_i^{\delta_i} = u_1^{\bar{a} \cdot \bar{d}_i} u_2^{\bar{b} \cdot \bar{d}_i} = \left[g_1^{x_1(\bar{a} \cdot \gamma^{(i)})} g_2^{x_2(\bar{b} \cdot \gamma^{(i)})} \right]^{\theta_i} = \left[g_1^{x_1 A} g_2^{x_2} \right]^{\theta_i(\bar{b} \cdot \gamma^{(i)})} = \left[g_1^{x_1 A} g_2^{x_2} \right]^{\bar{b} \cdot \bar{a}} = g_1^w = W$$

Therefore $\tilde{C} \in \mathsf{CT}_k(W)$. Next, we show that given I, W the vector $\tilde{C}$ is uniform in $\mathsf{CT}_k(W)$ over the choice of $\bar{a}, \bar{b}$. To do so we show that if $\tilde{C} = \langle S, g_1^{z_1}, \ldots, g_1^{z_{2k}} \rangle$ then $\bar{z} = \langle z_1, \ldots, z_{2k} \rangle$ is unform in a linear space of dimension k. Since $\mathsf{CT}_k(W)$ is associated to a linear space of dimension k and $\tilde{C} \in \mathsf{CT}_k(W)$ it will follow that $\tilde{C}$ spans all of $\mathsf{CT}_k(W)$ over the choice of $\bar{a}, \bar{b}$ and is thus uniform in it.

Let $\tilde{C} = \langle S, g_1^{z_1}, \ldots, g_1^{z_{2k}} \rangle$. Then by definition of $\tilde{C}$ we know that $\bar{z} = x_1 \bar{a} + c x_2 \bar{b}$. Therefore, it suffices to show that given I, W the vector $x_1 \bar{a} + c x_2 \bar{b}$ spans a space of dimension k over the choice of $\bar{a}, \bar{b}$. In Claim 4.3 we showed that $I = \langle y, h_1, \ldots, h_{2k}, \theta_1, \ldots, \theta_k \rangle$ induces $3k$ linear relations on the pair $\langle \bar{a}, \bar{b} \rangle$. Let Ω be the matrix as in Claim 4.3. We need to show that given that $\Omega \cdot [\bar{a} \| \bar{b}] = \bar{t}$, for some fixed $\bar{t} \in \mathbb{F}_q^{3k}$, the vector $x_1 \bar{a} + c x_2 \bar{b}$ spans a space of dimension k over the choice of $\bar{a}, \bar{b}$. This amounts to showing that the following $5k \times 4k$ matrix has full rank (i.e. has rank 4k):

$$\begin{pmatrix} \Gamma_0 & -A\Gamma_0 \\ I_{2k} & cI_{2k} \\ x_1 I_{2k} & c x_2 I_{2k} \end{pmatrix}$$

This is immediate since when $x_1 \neq x_2$ the two bottom parts of the matrix already have rank $4k$. This shows that given I, W the ciphertext $\tilde{C}$ comes from a linear space of dimension k. Hence, $\tilde{C}$ is unifrom in $\mathsf{CT}_k(W)$ over the choice of $\bar{a}, \bar{b}$. This concludes the proof that given I the pair $\langle W, \tilde{C} \rangle$ is distributed according to CW_k. □

Proof of Lemma 4.2: Claim 4.3 shows that in Step 8 of Algorithm $\mathcal{A}$ the pirate $\mathcal{P}$ is given input from a distribution that is identical to the distribution in the real attack. Consequently, it generates a pirate decoder $\mathcal{D}$ with the following properties. Let $\tilde{C} = \langle S, H_1, \ldots, H_{2k} \rangle$ for some $H_1, \ldots, H_{2k}$, let $W = \prod_{i=1}^{2k} H_i^{\alpha_i}$, and let $M = S/W$. When $\tilde{C}$ is a valid ciphertext then, by definition, M is the decryption of $\tilde{C}$. Given the pair $\langle \tilde{C}, M \rangle$ the decoder $\mathcal{D}$ satisfies: (1) if $\langle W, \tilde{C} \rangle$ is distributed according to CW_{valid} then the decoder returns "`valid`", and (2) if $\langle W, \tilde{C} \rangle$ is distributed according to CW_k it returns "`invalid`" with probability at least ϵ. By Claims 4.4 and 4.5 the pair $\langle W, \tilde{C} \rangle$ constructed in Step 9 falls into cases (1) or (2) depending on whether the given tuple $\langle g_1, g_2, u_1, u_2 \rangle$ is a Diffie-Hellman tuple. Hence, when the algorithm outputs its decision in Step 10 we have that:

$$\left| \Pr_{a,b \in \mathbb{F}_q} [\mathcal{A}(g, g^a, g^b, g^{ab}) = \text{"yes"}] - \Pr_{a,b,c \in \mathbb{F}_q} [\mathcal{A}(g, g^a, g^b, g^c) = \text{"yes"}] \right| \geq \epsilon$$

This concludes the proof of Lemma 4.2. □

4.5 Algorithm 1 Satisfies the Soundness Property

The next lemma shows that Algorithm 1 satisfies the soundness property (property 2). Recall that λ is the number of samples we make in Step 1 of Algorithm 1.

Lemma 4.6 *Let $\mathcal{D}$ be a pirate decoder built using a set $T_{\mathcal{D}}$ of at most k private keys. Let $T_{suspect}$ be a suspect set of size at most k. Let $\epsilon > 0$ and set $\lambda = 64k \log \frac{k}{\epsilon}$. Then when Algorithm 1 interacts with $\mathcal{D}$ and outputs "*`key` $\bar{d}$ `is guilty`*" then $\bar{d} \in T_{\mathcal{D}}$ with probability at least $1 - \epsilon$.*

Proof Since $\lambda = 64k \log \frac{k}{\epsilon}$ we know, by the Chernoff bound, that for all $i = 0, \ldots, k$ we have $|\hat{p}_i - P_{\mathcal{D},i}| < 1/8k$ with proability at least $1 - \epsilon$. Therefore, if key $\bar{d}_j \in T_{suspect}$ is pronounced guilty then with probability at least $1 - \epsilon$ we have

$$|P_{\mathcal{D},j} - P_{\mathcal{D},j-1}| \geq \frac{1}{4k}$$

This means that the pirate decoder is able to distinguish between the distributions CW_j and CW_{j-1} with advantage at least $\frac{1}{4k}$. The following lemma shows that if $\bar{d}_j \notin T_{\mathcal{D}}$ then this is not possible unless DDH in G_q cannot be solved with advantage $1/4k^2$. This will immediately prove Lemma 4.6

Lemma 4.7 *Let $\langle y, h_1, \ldots, h_{2k} \rangle$ be a public key. Let $T_{suspect} = \{\bar{d}_1, \ldots, \bar{d}_k\}$ be the set of k private keys used to define the sets CT_i for $i = 0, \ldots, k$. Suppose $\mathcal{P}$ is a pirate that given the public key and a random set of k private keys $T_{\mathcal{D}}$ is able to construct a decoder $\mathcal{D}$ with the following property:*

$$\exists j_0 \in \{1, \ldots, k\} : \quad \bar{d}_{j_0} \notin T_{\mathcal{D}} \quad \textit{and} \quad |P_{\mathcal{D},j_0} - P_{\mathcal{D},j_0-1}| \geq \epsilon$$

Then the pirate $\mathcal{P}$ can be used to solve DDH in G_q with advantage at least ϵ/k.

Proof Since $|P_{\mathcal{D},j_0} - P_{\mathcal{D},j_0-1}| \geq \epsilon$ we know that the pirate is able to build a decoder $\mathcal{D}$ that can distinguish CW_{j_0} from CW_{j_0-1} with advantage ϵ, without knowing $\bar{d}_j$. We show how to use such a pirate to solve DDH in G_q with advantage ϵ/k. Given a challenge tuple $\langle g_1, g_2, u_1, u_2 \rangle$ we use the following algorithm to decide whether it is a random tuple or a Diffie-Hellman tuple. We write $g_2 = g_1^c$ for some unknown c.

Algorithm $\mathcal{B}$:

Step 0: Pick a random $j_0 \in \{1, \ldots, k\}$.

Step 1: Choose a random set of k codewords in Γ. Denote these by $\gamma^{(1)}, \ldots, \gamma^{(k)} \in \mathbb{F}_q^{2k}$. (These will correspond to the k private keys given to the pirate.) Also, let $\gamma_s^{(1)}, \ldots, \gamma_s^{(k)}$ be an arbitrary set of k codewords in Γ. These will correspond to keys that belong to the suspect coalition. We assume $\gamma_s^{(j_0)} \notin \{\gamma^{(1)}, \ldots, \gamma^{(k)}\}$.

Step 2: Pick a random $A \in \mathbb{F}_q$ such that $g_1^A g_2 \neq 1$.

Step 3: Pick random vectors $\bar{a}, \bar{b} \in \mathbb{F}_q^{2k}$ such that the vector $\bar{a} - A\bar{b}$ is orthogonal to all vectors $\langle \gamma^{(1)}, \ldots, \gamma^{(k)}, \gamma_s^{(1)}, \ldots, \gamma_s^{(j_0-1)} \rangle$. If the chosen $\bar{a}, \bar{b}$ satisifes that $\bar{a} - A\bar{b}$ is orthogonal to $\gamma_s^{(j_0)}$ then repeat Step 3. Write $\bar{a} = (a_1, \ldots, a_{2k})$ and $\bar{b} = (b_1, \ldots, b_{2k})$.

Step 4: Set $h_i = g_1^{a_i} g_2^{b_i}$ for $i = 1, \ldots, 2k$.
Pick a random vector $\bar{v} \in \mathbb{F}_q^{2k}$ such that $\bar{v}$ is orthogonal to all $\gamma_s^{(1)}, \ldots, \gamma_s^{(j_0)}$.
Let $\bar{v} = \langle v_1, \ldots, v_{2k} \rangle$.

Step 5: Pick a random vector $\bar{\alpha} \in \mathbb{F}_q^{2k}$ such that $\bar{\alpha}$ is orthogonal to $\bar{a} - A\bar{b}$ and is orthogonal to $\bar{v}$.
Write $\bar{\alpha} = (\alpha_1, \ldots, \alpha_{2k})$.

Step 6: Set $y = h_1^{\alpha_1} \cdots h_{2k}^{\alpha_{2k}}$. Then $y = g_1^{\bar{\alpha} \cdot \bar{a}} g_2^{\bar{\alpha} \cdot \bar{b}} = g_1^{A(\bar{\alpha} \cdot \bar{b})} g_2^{\bar{\alpha} \cdot \bar{b}} = (g_1^A g_2)^{\bar{\alpha} \cdot \bar{b}}$.

Step 7: For $i = 1, \ldots, k$ set $\theta_i = \dfrac{\bar{\alpha} \cdot \bar{b}}{\gamma^{(i)} \cdot \bar{b}}$. Set $\bar{d}_i = \theta_i \gamma^{(i)}$ and set $T_{\mathcal{D}} = \{\bar{d}_1, \ldots, \bar{d}_k\}$.

For $i = 1, \ldots, j_0 - 1$ set $\theta_i^s = \dfrac{\bar{\alpha} \cdot \bar{b}}{\gamma_s^{(i)} \cdot \bar{b}}$. Set $\bar{d}_i^s = \theta_i^s \gamma_s^{(i)}$.

As in Algorithm $\mathcal{A}$ we know that $\bar{d}_1, \ldots, \bar{d}_k$ and $\bar{d}_1^s, \ldots, \bar{d}_{j_0-1}^s$ are representations of y to the base $h_1, \ldots, h_{2k}$.

Step 8: Run the pirate $\mathcal{P}$ giving it the public key $\langle y, h_1, \ldots, h_{2k} \rangle$ and the k private keys $\theta_1, \ldots, \theta_k$. The pirate builds a pirate decoder $\mathcal{D}$.

Step 9: Pick a random $S \in G_q$ and construct the ciphertext: $\tilde{C} = \langle S,\ g_1^{v_1} u_1^{a_1} u_2^{b_1},\ \ldots,\ g_1^{v_{2k}} u_1^{a_{2k}} u_2^{b_{2k}} \rangle$
Let $\tilde{C} = \langle S, H_1, \ldots, H_{2k} \rangle$ and set $W = \prod_{i=1}^{2k} H_i^{\alpha_i}$.

Step 10: Query the pirate decoder $\mathcal{D}$ on the ciphertext/message pair $\langle \tilde{C},\ S/W \rangle$. If $\mathcal{D}$ returns "valid" output that $\langle g_1, g_2, u_1, u_2 \rangle$ is a Diffie-Hellman tuple. Otherwise, output that $\langle g_1, g_2, u_1, u_2 \rangle$ is a random tuple.

This concludes the description of algorithm $\mathcal{B}$ for solving DDH. We show that algorithm $\mathcal{B}$ above correctly decides whether the given tuple $\langle g_1, g_2, u_1, u_2 \rangle$ is a DDH tuple. This follows from the following three claims.

Claim 4.8 *The distribution on the public key $\langle y, h_1, \ldots, h_{2k} \rangle$ and the k private keys $\theta_1, \ldots, \theta_k$ given to the pirate $\mathcal{P}$ in Step 8 is identical to the distribution generated by the real key generation algorithm.*

Proof The proof is identical to the proof of Claim 4.3 with minor changes. The only difference is that the pair $\langle \bar{a}, \bar{b} \rangle$ is subject to $k + j_0 - 1$ constraints in Step 3. Hence, given the public key $\langle \bar{a}, \bar{b} \rangle$ sits in a space of dimension $k - j_0 + 1$. □

Let $\langle y, h_1, \ldots, h_{2k} \rangle$ be the public key generated in Step 6. For $i = 1, \ldots, k$ define $\theta_i^s \in \mathbb{F}_p$ as the value that makes $\theta_i^s \gamma_s^{(i)}$ be a representation of y base $h_1, \ldots, h_{2k}$. For $i = 1, \ldots, j_0 - 1$ the value θ_i^s is computed in Step 7, but for $i = j_0, \ldots, k$ the value θ_i^s is unknown to Algorithm $\mathcal{B}$. For $i = 1, \ldots, k$ define $\bar{d}_i^s = \theta_i^s \gamma_s^{(i)}$. As above, $\bar{d}_i^s$ is unknown to Algorithm $\mathcal{B}$ for $i = j_0, \ldots, j$.

Claim 4.9 *Let $T_{suspect} = \{\bar{d}_1^s, \ldots, \bar{d}_k^s\}$ be the suspect set used to define the sets $\mathsf{CT}_0, \ldots, \mathsf{CT}_k$. Suppose $\langle g_1, g_2, u_1, u_2 \rangle$ is a random Diffie-Hellman tuple. As in Step 1 we assume $\bar{d}_{j_0}^s \notin T_{\mathcal{D}}$. Then given the public key $\langle y, h_1, \ldots, h_{2k} \rangle$ and the k private keys $\theta_1, \ldots, \theta_k$ the pair $\langle W, \tilde{C} \rangle$ generated in Step 9 is distributed according to CW_{j_0}.*

Proof We write $g_2 = g_1^c$ for some unknown c. We also let $u_1 = g_1^x$ and $u_2 = g_2^x$ for some unknown x. We need to show that the pair $\langle W, \tilde{C} \rangle$ generated in Step 9 satisfies: (1) W is uniform in G_q and independent of $I = \langle y, h_1, \ldots, h_{2k}, \theta_1, \ldots, \theta_k \rangle$, and (2) $\tilde{C}$ is uniform in $\mathsf{CT}_{j_0}(W)$ and independent of $\langle I, W \rangle$.

We start with W. Define $w = x(A + c)(\bar{b} \cdot \bar{\alpha})$. Then, since $\bar{\alpha}$ is orthogonal to $\bar{v}$ we have:

$$W = \prod_{i=1}^{2k} (g_1^{v_i} u_1^{a_i} u_2^{b_i})^{\alpha_i} = g_1^{\bar{v} \cdot \bar{\alpha}} u_1^{\bar{a} \cdot \bar{\alpha}} u_2^{\bar{b} \cdot \bar{\alpha}} = u_1^{A(\bar{b} \cdot \bar{\alpha})} u_2^{\bar{b} \cdot \bar{\alpha}} = g_1^{x(A+c)(\bar{b} \cdot \bar{\alpha})} = g_1^w$$

To see that given I the value W is uniform in G_q it suffices to show that w is uniform in $\mathbb{F}_q$. This is immediate since $\bar{\alpha} \cdot \bar{b} \neq 0$ (since $y \neq 1$), and $A + c \neq 0$ (since $g_1^A g_2 \neq 1$), and x is independent of I.

Next, we show that given I, W the ciphertext $\tilde{C}$ is uniform in $\mathsf{CT}_{j_0}(W)$. First, observe that $\tilde{C}$ generated in Step 9 satifies $\tilde{C} \in \mathsf{CT}_{j_0}(w)$. To see this write $\tilde{C} = \langle S, H_1, \ldots, H_{2k} \rangle$. Then $H_i = g_1^{v_i} u_1^{a_i} u_2^{b_i}$. Recall that $\bar{v}$ is chosen so that it is orthogonal to any $\bar{d}_i^s$ for $i = 1, \ldots, j_0$. Therefore, for any $i = 1, \ldots, j_0$

and $\bar{d}_i^s = (\delta_1, \ldots, \delta_{2k})$ we have:

$$\prod_{i=1}^{2k} H_i^{\delta_i} = g_1^{\bar{v}\cdot\bar{d}_i^s} u_1^{\bar{a}\cdot\bar{d}_i^s} u_2^{\bar{b}\cdot\bar{d}_i^s} = \left[g_1^{(\bar{a}\cdot\gamma_s^{(i)})} g_2^{(\bar{b}\cdot\gamma_s^{(i)})}\right]^{x\theta_i^s} = \left[g_1^{(\bar{a}+c\bar{b})\cdot\gamma_s^{(i)}}\right]^{x\theta_i^s} = \left[g_1^{(\bar{a}+c\bar{b})\cdot\bar{\alpha}}\right]^x = \left[g_1^{(A+c)(\bar{b}\cdot\bar{\alpha})}\right]^x = g_1^w = W$$

Therefore $\tilde{C} \in \mathsf{CT}_{j_0}(W)$. Next, we show that given I, W the vector $\tilde{C}$ is uniform in $\mathsf{CT}_{j_0}(W)$ over the choice of $\bar{v}$.

We first show that $\bar{v}$ is independent of I. Clearly $\bar{v}$ is independent of $h_1, \ldots, h_{2k}$. To show that it is independent of y we show that $\bar{v}$ is independent of $\bar{b}\cdot\bar{\alpha}$. That is,

$$\Pr_{\bar{v},\bar{\alpha}}\left[\bar{v} = \bar{v}_0 \mid \bar{\alpha}\cdot\bar{v}_0 = 0,\ \bar{\alpha}\cdot\bar{b} = t,\ \bar{\alpha}\cdot(\bar{a} - A\bar{b}) = 0\right] = \Pr_{\bar{v},\bar{\alpha}}[\bar{v} = \bar{v}_0]$$

For any $\bar{v}_0$ that is orthogonal to all $\gamma_s^{(1)}, \ldots, \gamma_s^{(j_0)}$ and $t \in \mathbb{F}_q$. This follows from the fact that $\bar{v}_0$ is linearly independent of $\bar{b}$ since $\bar{b}\cdot\gamma_s^{(1)} \neq 0$, and $\bar{v}_0$ is linearly independent of $\bar{a} - A\bar{b}$ since since $\bar{a} - A\bar{b}$ is not orthogonal to $\gamma_s^{(j_0)}$

Since $\bar{v}$ is independent of I, W, and $\bar{v}$ is uniform in a space of dimension $2k - j_0$, we know that $\tilde{C}$ is associated to a linear space of dimension $2k - j_0$ and is independent of I, W. But $2k - j_0$ is also the dimension of the linear space associated to $\mathsf{CT}_{j_0}(W)$. Hence, given I, W the ciphertext $\tilde{C}$ is uniform in $\mathsf{CT}_{j_0}(W)$. □

Claim 4.10 *Let $T_{suspect} = \{\bar{d}_1^s, \ldots, \bar{d}_k^s\}$ be the suspect set used to define the sets $\mathsf{CT}_0, \ldots, \mathsf{CT}_k$. Suppose $\langle g_1, g_2, u_1, u_2\rangle$ is a random tuple. As in Step 1 we assume $\bar{d}_{j_0}^s \notin T_{\mathcal{D}}$. Then given the public key $\langle y, h_1, \ldots, h_{2k}\rangle$ and the k private keys $\theta_1, \ldots, \theta_k$ the pair $\langle W, \tilde{C}\rangle$ generated in Step 9 is distributed according to CW_{j_0-1}.*

Proof We write $g_2 = g_1^c$ for some unknown c. We also let $u_1 = g_1^{x_1}$ and $u_2 = g_2^{x_2}$ for some unknown x_1, x_2. We assume $x_1 \neq x_2$. We need to show that the pair $\langle W, \tilde{C}\rangle$ generated in Step 9 satisfies: (1) W is uniform in G_q and independent of $I = \langle y, h_1, \ldots, h_{2k}, \theta_1, \ldots, \theta_k\rangle$, and (2) $\tilde{C}$ is uniform in $\mathsf{CT}_{j_0-1}(W)$ and independent of $\langle I, W\rangle$.

We start with W. Define $w = (x_1 A + cx_2)(\bar{\alpha}\cdot\bar{b})$. Then since $\bar{v}$ is orthogonal to $\bar{\alpha}$:

$$W = g_1^{\bar{v}\cdot\bar{\alpha}} u_1^{\bar{a}\cdot\bar{\alpha}} u_2^{\bar{b}\cdot\bar{\alpha}} = g_1^{x_1(\bar{a}\cdot\bar{\alpha})} g_2^{x_2(\bar{b}\cdot\bar{\alpha})} = \left[g_1^{x_1 A} g_2^{x_2}\right]^{(\bar{b}\cdot\bar{\alpha})} = g_1^w$$

To see that given I the value W is uniform in G_q it suffices to show that w is uniform in $\mathbb{F}_q$. This is immediate since $\bar{\alpha}\cdot\bar{b} \neq 0$ (since $y \neq 1$), and $Ax_1 + cx_2$ is independent of I.

Next, we show that given I, W the ciphertext $\tilde{C}$ is uniform in $\mathsf{CT}_{j_0-1}(W)$. First, observe that $\tilde{C}$ generated in Step 9 satifies $\tilde{C} \in \mathsf{CT}_{j_0-1}(W)$. To see this write $\tilde{C} = \langle S, H_1, \ldots, H_{2k}\rangle$. Then $H_i = g_1^{v_i} u_1^{a_i} u_2^{b_i}$. Recall that $\bar{v}$ is chosen so that it is orthogonal to any $\bar{d}_i^s$ for $i = 1, \ldots, j_0$. Therefore, for any $i = 1, \ldots, j_0 - 1$ and $\bar{d}_i^s = (\delta_1, \ldots, \delta_{2k})$ we have:

$$\prod_{i=1}^{2k} H_i^{\delta_i^s} = g_1^{\bar{v}\cdot\bar{d}_i^s} u_1^{\bar{a}\cdot\bar{d}_i^s} u_2^{\bar{b}\cdot\bar{d}_i^s} = \left[g_1^{x_1(\bar{a}\cdot\gamma_s^{(i)})} g_2^{x_2(\bar{b}\cdot\gamma_s^{(i)})}\right]^{\theta_i^s} = \left[g_1^{x_1 A} g_2^{x_2}\right]^{\theta_i^s(\bar{b}\cdot\gamma_s^{(i)})} = \left[g_1^{x_1 A} g_2^{x_2}\right]^{\bar{b}\cdot\bar{\alpha}} = g_1^w = W$$

Therefore $\tilde{C} \in \mathsf{CT}_{j_0-1}(W)$. Next, we show that given I, W the vector $\tilde{C}$ is uniform in $\mathsf{CT}_{j_0-1}(W)$. When $\tilde{C} = \langle S, g_1^{z_1}, \ldots, g_1^{z_{2k}}\rangle$ we know that $\bar{z} = \bar{v} + x_1\bar{a} + cx_2\bar{b}$. Hence, it suffices to show that given

I, W the vector $\bar{z}$ is uniform in a space of dimension $2k - j_0 + 1$ over the choice of $\bar{v}$ and x_1, x_2. As in Claim 4.9 we know that $\bar{v}$ is independent of I, W, and therefore it is uniform in a space of dimension $2k - j_0$. Hence, it suffices to show that given I, W the vector $x_1\bar{a} + cx_2\bar{b}$ is uniform in a space of dimension 1 over the choice of x_1, x_2 and it is linearly independent of the space of dimension $2k - j_0$ spanned by $\bar{v}$. First, since $x_1 \neq x_2$ we know that $x_1\bar{a} + cx_2\bar{b}$ is independent of $h_1, \ldots, h_{2k}$. However, the value w induces one affine constraint on x_1, x_2 since $x_1 A + cx_2 = w/(\bar{\alpha} \cdot \bar{b})$. Hence, given I, W the vector $x_1\bar{a} + cx_2\bar{b}$ spans an affine space of dimension 1 as required. Note that when $x_1 A + cx_2 = 0$ the space $x_1\bar{a} + cx_2\bar{b}$ is a linear space of dimension 1. We show that this space is not contained in the space spanned by $\bar{v}$ by showing that when $Ax_1 + cx_2 = 0$ the vector $x_1\bar{a} + cx_2\bar{b}$ cannot be orthogonal to $\gamma_s^{(j_0)}$:

$$(x_1\bar{a} + cx_2\bar{b}) \cdot \gamma_s^{(j_0)} = x_1(\bar{a} - A\bar{b}) \cdot \gamma_s^{(j_0)} + (cx_2 + Ax_1)(\bar{b} \cdot \gamma_s^{(j_0)}) = x_1(\bar{a} - A\bar{b}) \cdot \gamma_s^{(j_0)}$$

But since in Step 3 we make sure that $\bar{a} - A\bar{b}$ is not orthogonal to $\gamma_s^{(j_0)}$ we obtain the desired result. Hence, given I, W the ciphertext $\tilde{C}$ is uniform in $\mathsf{CT}_{j_0-1}(W)$. This shows that given I the pair $\langle W, \tilde{C} \rangle$ is chosen according to the distribution CW_{j_0-1}. □

Proof of Lemma 4.7: By Claim 4.8 the pirate is given input sampled from a distribution that is identical to the distribution in the real attack. Hence, it constructs a pirate decoder $\mathcal{D}$ satisfying the conditions of Lemma 4.7 for some $j_0 \in \{1, \ldots, k\}$. Then Claims 4.9 and 4.10 show that when j_0 is guessed correctly in Step 0 the pair $\langle W, \tilde{C} \rangle$ generated in Step 9 is distributed according to CW_{j_0} or CW_{j_0-1} depending on whether $\langle g_1, g_2, u_1, u_2 \rangle$ is a Diffie-Hellman tuple or a random tuple. Hence, when j_0 is guesses correctly, Algorithm $\mathcal{B}$ solves DDH in G_q with advantage ϵ. The probability that j_0 is guessed correctly is $1/k$, hence, Algorithm $\mathcal{B}$ solves DDH in G_q with advantage ϵ/k. □

5 Full Access Black box tracing against single-key pirates

Suppose a pirate obtains the private keys $\bar{d}_1, \ldots, \bar{d}_k$. A natural strategy for the pirate to build a pirate decryption box is to form a new random representation $\bar{d}$ and then create a box that decrypts using this representation. We call this a "single-key pirate" since only a single representation of y is embedded in the pirate decoder. We show an efficient full access black box tracing algorithm that works against a single-key pirate. Note that when a single user attempts to construct a decoder that cannot be traced back to her she is essentially acting as a single-key pirate.

Formally, we model the single-key pirate as two distinct parties $(\mathcal{P}_1, \mathcal{P}_2)$. The first party $\mathcal{P}_1$, called the *key-builder*, is given the public key, and k random private keys $\bar{d}_1, \ldots, \bar{d}_k$. It creates a new key $\bar{d}$ by forming a random convex combination of the given private keys. The key-builder then hands $\bar{d}$ to the second party $\mathcal{P}_2$, called the *box-builder*. The box-builder, seeing only $\bar{d}$ and the public key, is free to implement the pirate decryption box however it wants.

We show how the tracer can extract the representation $\bar{d} = (\delta_1, \ldots, \delta_{2k})$ from a pirate decoder created by a single-key pirate. Once $\bar{d}$ is obtained we use the algorithm of the Section 3.3 to recover the pirate keys. The basic idea in extracting $\bar{d}$ is to observe the decoder's behavior on *invalid ciphertexts*, e.g., $\tilde{C} = \langle S, h_1^{z_1}, \ldots, h_{2k}^{z_{2k}} \rangle$, where the non-constant vector $\bar{z}$ is chosen by the tracer. This ciphertext is invalid since the h_i's are raised to different powers. The next lemma shows that the pirate decoder cannot distinguish invalid ciphertexts from valid ciphertexts (assuming the difficulty of DDH in G_q).

Hence, we show that on input $\tilde{C} = \langle S, H_1, \ldots, H_{2k} \rangle$ it must respond with A where

$$A = S/\prod H_i^{\delta_i} = S/\prod h_i^{z_i \delta_i}$$

Non-traceable pirates: Let $(\mathcal{P}_1, \mathcal{P}_2)$ be a single-key pirate. Let $\langle y, h_1, \ldots, h_{2k} \rangle$ be a public key and let $\bar{d} = \{\delta_1, \ldots, \delta_{2k}\}$ be the private key generated by the key-builder $\mathcal{P}_1$ given the public key and a random set of k private keys. By assumption, $\bar{d}$ is a random convex combination of the k private keys. We say that the single-key pirate $(\mathcal{P}_1, \mathcal{P}_2)$ is *non-traceable* if given $\bar{d}$ the box builder $\mathcal{P}_2$ is able to create a pirate decoder $\mathcal{D}$ with the following properties: (1) $\mathcal{D}$ correctly decrypts all valid ciphertexts, and (2) when $\mathcal{D}$ is given a random invalid ciphertext $\tilde{C} = \langle S, H_1, \ldots, H_{2k} \rangle \in G_q^{2k+1}$ it outputs a value different from $S/\prod H_i^{\delta_i}$, with non-negligible probability (over the choice of $\tilde{C}$). The following lemma shows that a non-traceable single-key pirate does not exist.

Lemma 5.1 *Suppose $(\mathcal{P}_1, \mathcal{P}_2)$ is a non-traceable single-key pirate. Then $(\mathcal{P}_1, \mathcal{P}_2)$ can be used to solve DDH in G_q.*

Proof Sketch Given a challenge tuple $\langle g_1, g_2, u_1, u_2 \rangle$ we decide whether it is a random tuple or a Diffie-Hellman tuple as follows:

Step 1: Pick a random subset of k vectors $\gamma^{(1)}, \ldots, \gamma^{(k)} \subseteq \Gamma$ and create a random linear combination $\hat{\gamma} \in \mathbb{F}_q^{2k}$ of these k vectors. The vector $\hat{\gamma}$ will eventually become the private key created by the key-builder $\mathcal{P}_2$.

Step 2: Pick random vectors $\bar{a}, \bar{b} \in \mathbb{F}_q^{2k}$ such that $(\bar{a} - \bar{b}) \cdot \hat{\gamma} = 0$.

Step 3: Set $h_i = g_1^{a_i} g_2^{b_i}$ for $i = 1, \ldots, 2k$.

Step 4: Pick a random vector $\bar{\alpha} \in \mathbb{F}_q^{2k}$ such that $\bar{\alpha} \cdot (\bar{a} - \bar{b}) = 0$.

Step 5: Set $y = \prod h_i^{\alpha_i}$.

Step 6: Set $\hat{\theta} = \frac{\bar{\alpha} \cdot \bar{b}}{\hat{\gamma} \cdot \bar{b}}$.

Set $\bar{d} = \hat{\theta}\hat{\gamma} = \langle \delta_1, \ldots, \delta_{2k} \rangle$. Observe that $\bar{d}$ is a representation of y base $h_1, \ldots, h_{2k}$.

Step 7: Run the box builder $\mathcal{P}_2$ giving it the public key $\langle y, h_1, \ldots, h_{2k} \rangle$ and the representation $\bar{d}$. The box builder builds a pirate decoder $\mathcal{D}$.

Step 8: Using a random $S \in G_q$ construct the ciphertext: $\tilde{C} = \langle S,\ u_1^{a_1} u_2^{b_1},\ \ldots,\ u^{a_{2k}} v^{b_{2k}} \rangle$.
Let $\tilde{C} = \langle S, H_1, \ldots, H_{2k} \rangle$.

Step 9: Query the pirate decoder $\mathcal{D}$ on the ciphertext $\tilde{C}$. If $\mathcal{D}$ returns $S/\prod_{i=1}^{2k} H_i^{\alpha_i}$ output that $\langle g_1, g_2, u_1, u_2 \rangle$ is a Diffie-Hellman tuple. Otherwise, output $\langle g_1, g_2, u_1, u_2 \rangle$ is a random tuple.

Observe that the input $I = \langle y, h_1, \ldots, h_{2k}, \bar{d} \rangle$ given to the box-builer $\mathcal{P}_2$ is sampled from a distributed identical to the real attack. To see this note that $y, h_1, \ldots, h_{2k}$ are uniformly distributed in G_q^{2k}. Furthermore, $\bar{d}$ is a random convex combination of a random set of k private keys.

Next, note that when the challenge $\langle g_1, g_2, u_1, u_2 \rangle$ is a Diffie-Hellman tuple then given I, the ciphertext $\tilde{C}$ is a random valid ciphertext. Otherwise, given I the ciphertext $\tilde{C}$ is uniformly distributed in G_q^{2k+1}. Since the decoder behaves differently for valid and invalid ciphertexts the output in Step 9 enables us to solve the given DDH challenge. □

By querying at invalid ciphertexts the tracer learns the value $\prod h_i^{z_i \delta_i} = S/A$ for vectors $\bar{z}$ of its choice. After $2k$ queries with random linearly independent $\bar{z}$, the tracer can solve for $h_1^{\delta_1}, \ldots, h_{2k}^{\delta_{2k}}$. Since the tracer knows the discrete log of the h_i's base g (recall Step 2 of key generation) it can compute $g^{\delta_1}, \ldots, g^{\delta_{2k}}$. Ideally, we would like to use homomorphic properties of the discrete log to run the tracing algorithm of the previous section "in the exponents". Unfortunately, it is an open

problem to run Berlekamp's algorithm this way. Instead, we can recover the vector $\bar{d} = (\delta_1, \ldots, \delta_{2k})$ from $\langle g^{\delta_1}, \ldots, g^{\delta_{2k}} \rangle$ by using recent results on trapdoors of the discrete log [12, 14] modulo p^2q and modulo N^2. For instance, the trapdoor designed by Paillier [14] shows that if encryption is done in the group $\mathbb{F}^*_{N^2}$ then the secret factorization of N (known to the tracer only) enables the tracer to recover $\langle \delta_1, \ldots, \delta_{2k} \rangle$ mod N from $\langle g^{\delta_1}, \ldots, g^{\delta_{2k}} \rangle$. The tracing algorithm of the previous section can now be used to recover the keys at the pirate's possession. This completes the description of the black box tracing algorithm for single-key pirates.

6 Chosen ciphertext security

In a typical scenario where our system is used it is desirable to defend against chosen ciphertext attacks. Fortunately, our scheme can be easily modified to be secure against adaptive attacks. The modification is similar to the approach used by Cramer and Shoup [6]. As in Section 3 we work in a group G_q of prime order q. For example, G_q could be a subgroup of order q of $\mathbb{F}^*_p$ for some prime p where $q|p-1$.

Key generation: Let g be a generator of G_q. Pick random $r_1, \ldots, r_{2k} \in \mathbb{F}_q$ and set $h_i = g^{r_i}$ for $i = 1, \ldots, 2k$. Next, we pick random $x_1, x_2, y_1, y_2 \in \mathbb{F}_q$ and $\alpha_1, \ldots, \alpha_{2k} \in \mathbb{F}_q$ and compute

$$y = h_1^{\alpha_1} h_2^{\alpha_2} \cdots h_{2k}^{\alpha_{2k}} \; ; \quad c = h_1^{x_1} h_2^{x_2} \; ; \quad d = h_1^{y_1} h_2^{y_2}$$

The public key is $\langle y, c, d, h_1, \ldots, h_{2k} \rangle$. The private key is as in Section 3, but also includes $\langle x_1, x_2, y_1, y_2 \rangle$. Hence, user i's private key is $\langle \theta_i, x_1, x_2, y_1, y_2 \rangle$.

Encryption: To encrypt a message $M \in G_q$ do the following: pick a random element $a \in \mathbb{F}_q$, and compute

$$S = M \cdot y^a \; ; \quad H_1 = h_1^a \; , \; \ldots \; , \; H_{2k} = h_{2k}^a$$
$$\nu = \mathcal{H}(S, H_1, \ldots, H_{2k}) \; ; \quad v = c^a d^{a\nu}$$

where $\mathcal{H}$ is a collision resistant hash function (or chosen from a family of universal one-way hash functions). Set the ciphertext C to be

$$C = \langle S, H_1, \ldots, H_{2k}, v \rangle$$

It is a bit surprising that the system can be made secure against chosen ciphertext attacks by appending a single element v to the ciphertext.

Decryption: To decrypt a ciphertext $C = \langle S, H_1, \ldots, H_{2k}, v \rangle$ using a private key $\langle \theta_i, x_1, x_2, y_1, y_2 \rangle$ first compute $\nu = \mathcal{H}(S, H_1, \ldots, H_{2k})$ and check that

$$H_1^{x_1 + y_1 \nu} \cdot H_2^{x_2 + y_2 \nu} = v$$

If the test fails, reject the ciphertext. Otherwise, output

$$M = S / U^{\theta_i} \qquad \text{where} \qquad U = \prod_{j=1}^{2k} H_j^{\gamma_j}$$

and $\gamma^{(i)} = (\gamma_1, \ldots, \gamma_{2k}) \in \Gamma$ is the codeword from which θ_i is derived.

Tracing: The tracing algorithm remains unchanged.

We show that the scheme is secure against adaptive chosen ciphertext attack. In other words, we show that the scheme is secure in the following environment: an adversary is given the public key. It generates two messages M_0, M_1 and is given the encryption $C = E(M_b)$ for $b \in \{0,1\}$ chosen at random. The adversary's goal is to predict b. To do so he is allowed to interact with a decryption oracle that will decrypt any valid ciphertext other than C. If the adversary's guess for b is b' and the probability that $b = b'$ is $\frac{1}{2} + \epsilon$ then we say that the adversary has advantage ϵ. The system is said to be secure against an adaptive chosen ciphertext attack if the adversary's advantage in predicting b is negligible (as a function of the security parameter).

Theorem 6.1 *The above cryptosystem is secure against an adaptive chosen ciphertext attack assuming that (1) the Decision Diffie-Hellman problem is hard in the group G_q, and (2) the hash function $\mathcal{H}$ is collision resistant (or chosen from a family of universal one-way hash functions).*

We assume the hash function $\mathcal{H}$ is collision resistant. Suppose there exists a polynomial time adversary $\mathcal{A}$ that is able to obtain a non-negligible advantage in predicting b when the above cryptosystem is used. We show that $\mathcal{A}$ can be used to solve the Decision Diffie-Hellman problem in G_q.

Given a tuple $\langle g_1, g_2, u_1, u_2 \rangle$ in G_q we perform the following steps to determine if it is a random tuple (i.e chosen from R) or a Diffie-Hellman tuple (i.e chosen from D):

Init Set $h_1 = g_1$ and $h_2 = g_2$. pick random $r_3, \ldots, r_{2k} \in \mathbb{F}_q$ and set $h_i = g_2^{r_i}$ for $i = 3, \ldots, 2k$. Next, choose random $x_1, x_2, y_1, y_2 \in \mathbb{F}_q$ and $\alpha_1, \ldots, \alpha_{2k} \in \mathbb{F}_q$ and compute

$$y = h_1^{\alpha_1} \cdots h_{2k}^{\alpha_{2k}} \; ; \quad c = h_1^{x_1} h_2^{x_2} \; ; \quad d = h_1^{y_1} h_2^{y_2}$$

Challenge The adversary $\mathcal{A}$ is given the public key and outputs two messages $M_0, M_1 \in G_q$. We pick a random $b \in \{0,1\}$ and compute:

$$S = M_b \cdot u_1^{\alpha_1} u_2^{\alpha_2} \prod_{i=3}^{2k} u_2^{\alpha_i r_i}$$

$$H_1 = u_1 \, , \quad H_2 = u_2 \, , \quad H_3 = u_2^{r_3} \, , \quad \ldots \, , \quad H_{2k} = u_2^{r_{2k}}$$

$$\nu = \mathcal{H}(S, H_1, \ldots, H_{2k}) \; ; \qquad v = u_1^{x_1 + y_1 \nu} u_2^{x_2 + y_2 \nu}$$

The challenge ciphertext given to $\mathcal{A}$ is $C = \langle S, H_1, \ldots, H_{2k}, v \rangle$.

Interaction When the adversary $\mathcal{A}$ asks to decrypt a ciphertext

$$C' = \langle S', H_1', \ldots, H_{2k}', v' \rangle$$

we respond as in a normal decryption: first we check validity of the ciphertext and reject invalid ciphertexts. For valid ciphertext we give $\mathcal{A}$ the plaintext $M = S' / \prod_{j=1}^{2k} (H_i')^{\alpha_j}$.

Output Eventually the adversary $\mathcal{A}$ outputs a $b' \in \{0,1\}$. If $b = b'$ we say the input tuple is from D otherwise we say R.

This completes the description of the algorithm for deciding DDH using $\mathcal{A}$. To complete the proof of Theorem 6.1 it remains to show two things:

- When $\langle g_1, g_2, u_1, u_2 \rangle$ is chosen from D the joint distribution of the adversary's view and the bit b is statistically indistinguishable from the actual attack.

- When $\langle g_1, g_2, u_1, u_2 \rangle$ is chosen from R the hidden bit b is (essentially) independent of the adversary's view.

The proofs of both statements are similar to the proofs given by Cramer and Shoup [6] and is given in the appendix. Based on the two statements a standard argument shows that if the adversary $\mathcal{A}$ has advantage ϵ in predicting b then the above algorithm for deciding DDH also has advantage ϵ. This completes the proof of Theorem 6.1.

Key extraction and black box tracing. It is surprising that although the scheme is resistant to chosen ciphertext attack, the decryption box will decrypt invalid ciphertexts. In particular, it will decrypt an invalid ciphertext $\tilde{C} = \langle S, H_1, \ldots, H_{2k}, v \rangle$ where

$$S = M \cdot y^a \; ; \quad H_1 = h_1^a \; , \quad H_2 = h_2^a \; , \quad H_3 = h_3^{b_3} \; , \quad H_4 = h_4^{b_4} \; , \ldots \; , \quad H_{2k} = h_{2k}^{b_{2k}}$$

$$\nu = \mathcal{H}(S, H_1, \ldots, H_{2k}) \; ; \quad v = c^a d^{a\nu}$$

This is an invalid ciphertext since the h_i's are raised to different powers. It passes the decryptor's test since h_1 and h_2 are raised to the same power. It cannot be distinguished from a valid ciphertext, assuming the hardness of DDH in G_q. The ideas of Section 4 can then be applied for black box tracing and black box confirmation in this setting. For example, suppose a "single-key pirate" constructs a pirate decoder containing the representation $\bar{d} = (d_1, \ldots, d_{2k})$. By feeding the decoder invalid ciphertexts as above the tracer can recover $\langle g^{d_3}, \ldots, g^{d_{2k}} \rangle$. Using trapdoors of the discrete log the tracer can find $\langle d_3, \ldots, d_{2k} \rangle$ from which he can expose all active members of a coalition of size at most $k-1$. As in section 4 black box tracing is possible against any pirate by using "black box confirmation".

7 An open problem

The black box confirmation method of Section 4 gives rise to a very general black box tracing strategy. The tracing strategy works even when the pirate box has $\ell - 1$ out of the total of ℓ keys. Let $d_1, \ldots, d_\ell$ be the set of all ℓ private keys generated by the key generation algorithm. Let T_i be the set of keys $\{d_1, ..., d_i\}$. There are ℓ such sets with T_0 being the empty set and T_ℓ being the set of all private keys. Consider an encryption scheme with the following properties:

- Given a message M and an $i \in \{0, \ldots, \ell\}$ there is a set of ciphertexts CT_i such that $C \in CT_i$ decrypts correctly to M using all private keys in T_i, but decrypts to some $M' \neq M$ when using keys not in T_i (there could be a different M' for each $d \notin T_i$). Note that all of CT_ℓ are valid ciphertexts since they decrypt correctly to M under all private keys.
- An adversary given all ℓ private keys except for key d_j cannot tell whether a ciphertext C decrypts to M using key d_j.

Any ecryption scheme satisying both properties above gives rise to an efficient black box tracing algorithm. Let $CT_i(M)$ be the set of ciphertexts that decrypt to M using any key in T_i. Let CT_i be the union of $CT_i(M)$ over all M. Consider a pirate box that plays music. Define

$$p_i = \Pr_{x \in CT_i}[\text{pirate box plays music when given } x]$$

We know that $p_n = 1$ since CT_n is the set of valid ciphertexts. Similarly, we know $p_0 = 0$ since no key can decrypt messages in CT_0. Hence, since $|p_n - p_0| = 1$, there must exist a j such that $|p_j - p_{j+1}| > 1/\ell$. The only difference between CT_j and CT_{j+1} is that key d_{j+1} can decrypt ciphertexts in CT_{j+1} but cannot decrypt ciphertexts in CT_j. Hence, the pirate box is able to distinguish whether key d_j correctly decrypts the given ciphertext. By property (2) this is possible only if the pirate had d_j when building the pirate box. Hence, d_j can be pronounced guilty.

To summarize, the tracing algorithm works as follows:

Step 1: For $i = 0, ..., \ell$ estimate the value of p_i. This is done by picking many random elements in $C \in CT_i$ and testing whether the pirate box "plays music" when given C. Using $O(\ell)$ samples will produce an approximation $\hat{p}_i$ to p_i with error less than $1/4\ell$ with high probability.

Step 2: Find a j such that $|\hat{p}_j - \hat{p}_{j+1}| > 1/8\ell$. Output "$d_j$ `is guilty`" and stop.

The tracing algorithm makes a total of $O(\ell^2 \log \ell)$ probes into the pirate box. In an encryption scheme satisfying both properties above each message M can be encrypted in at least $\ell + 1$ different ways $(C_0, ..., C_\ell)$. Hence, the ciphertext length must be at least $\log_2 \ell$. Unfortunately, the current best constructions require a ciphertext whose length is linear in the number of users ℓ.

Open problem: Construct a secure encryption scheme satisfying both properties above where the ciphertext length is sub-linear in ℓ. Ideally the ciphertext length should be $O(\log \ell)$.

8 Conclusion

We present an efficient public key solution to the traitor tracing problem. Our construction is based on Reed-Solomon codes and the representation problem for discrete logs. Traceability follows from the hardness of discrete log. The semantic security of the encryption scheme against a passive attack follows from the Decision Diffie-Hellman assumption. A simple extension achieves security against an adaptive chosen ciphertext attack under the same hardness assumption. The private key in all cases is just a single element of a finite field and can be as short as 160 bits. The cryptosystem can be made to work in any group in which the Decision Diffie-Hellman problem is hard. It is an interesting open question to improve on the "black box" traceability of our approach. Also, it seems reasonable to believe that there exists an efficient public key traitor tracing scheme that is completely collusion resistant. In such a scheme, any number of private keys cannot be combined to form a new key. Similarly, the complexity of encryption and decryption is independent of the size of the coalition under the pirate's control. An efficient construction for such a scheme will provide a useful solution to the public key traitor tracing problem.

To conclude, we mention an application of our system to defending against software piracy. Typically, when new software is installed from a CD-ROM the user is asked to enter a short unique key printed on the CD cover. This key identifies the installed copy. Clearly the key printed on the CD cover has to be short (say under 20 characters) since it is typed in manually. Our system can be used in this settings as follows: since our private key can be made 120 bits long (to achieve 2^{60} security) it can be printed on the CD cover (each character encodes 6 bits). The software on the CD is encrypted using our system's public key. When the user types in his unique CD key the software is decrypted and installed on the user's machine. However, if a software pirate attempts to create illegal copies of the distribution CD (say using a CD-ROM burner) he must also attach a short printed key to the disk. Using our system, the key he attaches to the bootlegged copies can be traced back to him.

References

[1] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography: The case of hashing and signing", in proc. Crypto '94, 216–233.

[2] L. Welch and E. Berlekamp, "Error correction of algebraic block codes", U.S. Patent No. 4,633,470, issued December 1986.

[3] D. Boneh, "The decision Diffie-Hellman problem", In proc. of the Third Algorithmic Number Theory Symposium (ANTS), LNCS, Vol. 1423, Springer-Verlag, pp. 48–63, 1998.

[4] D. Boneh, M. Franklin, "An efficient public key traitor tracing scheme", In proc. of Crypto '99, LNCS, Vol. 1666, Springer-Verlag, pp. 338–353, 1999.

[5] B. Chor, A. Fiat and M. Naor, "Tracing traitors", in proc. Crypto '94, pp. 257–270.

[6] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack", in proc. Crypto '98, pp. 13–25.

[7] C. Dwork, J. Lotspiech and M. Naor, "Digital signets: self-enforcing protection of digital information", in proc. of STOC '96, pp. 489–498, 1996.

[8] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometric codes" in proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS), 1998.

[9] K. Kurosawa, and Y. Desmedt, "Optimum traitor tracing and asymmetric schemes", in proc. of Eurocrypt '98, pp. 145–157.

[10] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[11] M. Naor and B. Pinkas, "Threshold traitor tracing", in proc. Crypto '98, pp. 502–517.

[12] T. Okamoto, S. Uchiyama, "A new public key cryptosystem as secure as factoring", in proc. Eurocrypt '98, pp. 308–318.

[13] V. Pan, "Faster solution of the key equation for decoding BCH error-correcting codes" in proc. 29th ACM Symposium on Theory of Computation, pp. 168–175, 1997.

[14] P. Paillier, "Public-Key Cryptosystems Based on Discrete Logarithm Residues", in proc. Eurocrypt '99, pp. 223–238.

[15] B. Pfitzmann, "Trials of traced traitors", in proc. Information Hiding Workshop, pp. 49–64, 1996.

[16] B. Pfitzmann and M. Waidner, "Asymmetric fingerprinting for larger collusions", in proc. ACM Conference on Computer and Communication Security, pp. 151–160, 1997.

[17] D. Stinson and R. Wei, "Combinatorial properties and constructions of traceability schemes and frameproof codes", *SIAM Journal on Discrete Math*, 11(1), pp. 41–53, 1998.

[18] D. Stinson and R. Wei, "Key preassigned traceability schemes for broadcast encryption", in proc. SAC '98, LNCS 1556, 1999.

A Proof of Theorem 6.1

To complete the proof of Theorem 6.1 we need to prove the following lemma regarding the algorithm for deciding DDH in G_q.

Lemma A.1

a. *When $\langle g_1, g_2, u_1, u_2\rangle$ is chosen from D the joint distribution of the adversary's view and the bit b is statistically indistinguishable from the actual attack.*

b. *When $\langle g_1, g_2, u_1, u_2\rangle$ is chosen from R the hidden bit b is (essentially) independent of the adversary's view.*

Proof The proof is very similar to the one given by Cramer and Shoup [6]. Recall that the challenge ciphertext given to the adversary is $C = \langle S, H_1, \ldots, H_{2k}, v\rangle$ where

$$S = M_b \cdot u_1^{\alpha_1} u_2^{\alpha_2} \prod_{i=3}^{2k} u_2^{\alpha_i r_i} \; ; \quad H_1 = u_1 \, , \quad H_2 = u_2 \, , \quad H_3 = u_2^{r_3} \, , \quad \ldots \, , \quad H_{2k} = u_2^{r_{2k}}$$

$$\nu = \mathcal{H}(S, H_1, \ldots, H_{2k}) \; ; \quad v = u_1^{x_1 + y_1 \nu} u_2^{x_2 + y_2 \nu}$$

We start by proving part (a). Say $\langle g_1, g_2, u_1, u_2\rangle$ is chosen from D. Then there exists an a such that $u_1 = g_1^a$ and $u_2 = g_2^a$. In this case, the challenge ciphertext C given to the adversary is a valid encryption of M_b. Indeed, one can easily verify that the challenge ciphertext C satisfies:

$$\begin{aligned} S &= y^a \\ H_i &= h_i^a \quad \text{for} \quad i = 1, \ldots, 2k \\ v &= c^a d^{\nu a} \end{aligned}$$

Furthermore, during the interaction phase with the adversary $\mathcal{A}$ the decryption oracle given to $\mathcal{A}$ behaves exactly as it does in the actual attack.

The proof of part (b) is a bit harder. Say $\langle g_1, g_2, u_1, u_2\rangle$ is chosen from R. Let $u_1 = g_1^{a_1}$ and $u_2 = g_2^{a_2}$. With high probability $a_1 \neq a_2$. Also, let $g_1 = g_2^w$ (recall $h_1 = g_1$ and $h_2 = g_2$).

We say that a ciphertext $C = \langle S, H_1, \ldots, H_{2k}, v\rangle$ is *valid* if $\log_{g_1} H_1 = \log_{g_2} H_2$. The proof of part (b) follows from the following two claims:

Claim A.2 *If the decryption oracle given to the adversary $\mathcal{A}$ rejects all invalid ciphertexts then the distribution of the bit b is independent of the adversary's view.*

Proof The value of S in the challenge ciphertext is $S = M_b \cdot u_1^{\alpha_1} u_2^{\alpha_2} \prod_{i=3}^{2k} u_2^{\alpha_i r_i}$. We show that the adversary has no information about the blinding factor B where

$$B = u_1^{\alpha_1} u_2^{\alpha_2} \prod_{i=3}^{2k} u_2^{\alpha_i r_i}$$

Consequently, the adversary obtains no information about M_b – the blinding factor B is a perfect one time pad.

Consider the point $Q = (\alpha_1, \ldots, \alpha_{2k})$. Given the public key the adversary knows that Q is a random point satisfying:

$$\log_{g_2} y = w\alpha_1 + \alpha_2 + \sum_{i=3}^{2k} r_i \alpha_i \pmod{q} \tag{2}$$

When the decryption oracle decrypts a ciphertext $C' = \langle S', H'_1, \ldots, H'_{2k}, v' \rangle$ it gives the adversary the value $M' = S'/(H'_1)^{\alpha_1} \cdots (H'_{2k})^{\alpha_{2k}}$. Hence, the adversary learns that the point Q satisfies the following relation:

$$\log_{g_2} \frac{S'}{M'} = \alpha_1 \log_{g_2} H'_1 + \alpha_2 \log_{g_2} H'_2 + \ldots + \alpha_{2k} \log_{g_2} H'_{2k} \tag{3}$$

Since the decryption oracle decrypts only valid ciphertexts we know that $\log_{g_1} H'_1 = \log_{g_2} H'_2$, or equivalently $\log_{g_2} H'_1 = w \log_{g_2} H'_2$. If we write $s = \log_{g_2} H'_2$ then equation (3) becomes:

$$\log_{g_2} \frac{S'}{M'} = (ws)\alpha_1 + s\alpha_2 + \alpha_3 \log_{g_2} H'_3 + \ldots + \alpha_{2k} \log_{g_2} H'_{2k} \tag{4}$$

Observe that equation (2) and all of the equations (4) obtained by the adversary are restricted to a subspace of dimension $2k - 1$ (the coefficient of α_1 is always w times the coefficient of α_2).

To complete the proof of the claim note that the blinding factor B in the challenge ciphertext satisfies:

$$\log_{g_2} B = (wa_1)\alpha_1 + a_2\alpha_2 + \sum_{i=3}^{2k} (a_2 r_i)\alpha_i$$

Assuming $a_1 \neq a_2$ this relation on Q is linearly independent of equation (2) and all the equations (4). Hence, conditioned on the adversary's view, the blinding factor B is uniformly distributed in $\mathbb{F}_q$. It follows that the adversary has no information about M_b implying that its output b' is independent of b. □

Claim A.3 *The decryption oracle we supply to the adversary $\mathcal{A}$ will reject all invalid ciphertexts, except with negligible probability.*

Proof The proof is based on the fact that the adversary does not know the point $P = (x_1, x_2, y_1, y_2)$. From the public key and the challenge ciphertext the adversary knows that the point P satisfies the following relations:

$$\log_{g_2} c = x_1 w + x_2 \tag{5}$$

$$\log_{g_2} d = y_1 w + y_2 \tag{6}$$

$$\log_{g_2} v = (x_1 + y_1\nu)wa_1 + (x_2 + y_2\nu)a_2 \tag{7}$$

Suppose the adversary queries the decryption oracle at an invalid ciphertext $C' = \langle S', H'_1, \ldots, H'_{2k}, v' \rangle$ where $H'_1 = g_1^{b_1}$ and $H'_i = g_2^{b_i}$ for $i = 2, \ldots, 2k$. Let $\nu' = \mathcal{H}(S', H'_1, \ldots, H'_{2k})$. There are three cases to consider:

Case 1: $\langle S, H_1, \ldots, H_{2k} \rangle = \langle S', H'_1, \ldots, H'_{2k} \rangle$. Since $C \neq C'$ we have $v \neq v'$. Since v' is invalid the decryption oracle will reject C'.

Case 2: $\langle S, H_1, \ldots, H_{2k} \rangle \neq \langle S', H'_1, \ldots, H'_{2k} \rangle$ and $\nu = \nu'$. This implies the adversary found a collision in the hash function $\mathcal{H}()$, contradicting the assumption on $\mathcal{H}()$.

Case 3: $\langle S, H_1, \ldots, H_{2k} \rangle \neq \langle S', H'_1, \ldots, H'_{2k} \rangle$ and $\nu \neq \nu'$. The decryption oracle will reject the ciphertext, unless

$$H_1^{x_1+y_1\nu'} \cdot H_2^{x_2+y_2\nu'} = v'$$

In other words,

$$\log_{g_2} v' = wb_1(x_1 + y_1\nu') + b_2(x_2 + y_2\nu') \tag{8}$$

However, Equations (5),(6),(7) and (8) are linearly independent. This can be seen from the determinant of

$$\det \begin{pmatrix} w & 1 & 0 & 0 \\ 0 & 0 & w & 1 \\ wa_1 & a_2 & wa_1\nu & a_2\nu \\ wb_1 & b_2 & wb_1\nu' & b_2\nu' \end{pmatrix} = w^2(\nu - \nu')(b_1 - b_2)(a_1 - a_2)$$

Since the ciphertext is invalid we know that $b_1 \neq b_2$ and hence the determinant is non zero. Equation (5), (6) and (7) tell the adversary that P lies on a certain line $\mathcal{L}$. Hence, P is one of q possible points. By linear independence, the hyperplane defined by (8) intersects the line $\mathcal{L}$ at a point. For the adversary's first query the probability (over the choice of P) that the intersection occurs at the point P is $1/q$. Hence, the invalid ciphertext will be accepted by the decryption oracle with probability at most $1/q$. After the first query is rejected the adversary knows that the point P is now one of only $q-1$ different values along $\mathcal{L}$. Hence, for the second query, the probability that $\mathcal{L}$ intersects equation (8) at P is at most $\frac{1}{q-1}$. For the i'th query the probability is at most $\frac{1}{q-i}$. Overall, the probability that an invalid ciphertext is ever accepted is at most $\sum_{i=1}^{n} \frac{1}{q-i} \leq \frac{n}{q-n}$ where n is the total number of queries. Since the adversary runs in polynomial time q is exponential in n. Hence, the probability that an invalid ciphertext is ever accepted is negligible. □

Classification of Security Properties *
(Part I: Information Flow)

Riccardo Focardi[1] and Roberto Gorrieri[2]

[1] Dipartimento di Informatica
Università Ca'Foscari di Venezia
e-mail: focardi@dsi.unive.it
[2] Dipartimento di Scienze dell'informazione
Università di Bologna
e-mail: gorrieri@cs.unibo.it

Abstract. In the recent years, many formalizations of security properties have been proposed, most of which are based on different underlying models and are consequently difficult to compare. A classification of security properties is thus of interest for understanding the relationships among different definitions and for evaluating the relative merits. In this paper, many non-interference-like properties proposed for computer security are classified and compared in a unifying framework. The resulting taxonomy is evaluated through some case studies of access control in computer systems. The approach has been mechanized, resulting in the tool CoSeC. Various extensions (e.g., the application to cryptographic protocol analysis) and open problems are discussed.

This paper mainly follows [21] and covers the first part of the course "Classification of Security Properties" given by Roberto Gorrieri and Riccardo Focardi at FOSAD'00 school.

1 Introduction

The wide spread of distributed systems, where resources and data are shared among users located almost everywhere in the world, has enormously increased the interest in security issues. In this context, it is likely that a user gets some (possibly) malicious programs from an untrusted source on the net and executes them inside its own system with unpredictable results. Moreover, it could be the case that a system completely secure inside, results to be insecure when performing critical activities such as electronic commerce or home banking, due to a "weak" mechanism for remote connections. It is important to precisely define security properties in order to have formal statements of the correctness of a security mechanism. As a consequence, in the recent years there have been a

* This work has been partially supported by MURST projects TOSCA, "Certificazione automatica di programmi mediante interpretazione astratta" and "Interpretazione astratta, type systems e analisi control-flow", and also partially supported by Microsoft Research Europe.

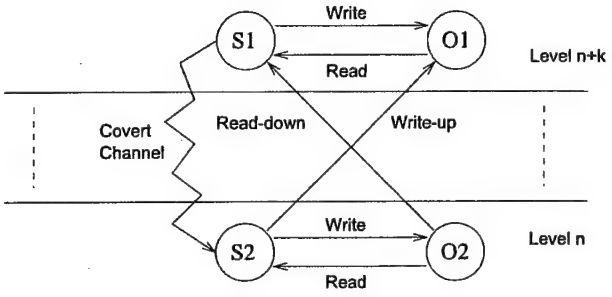


Fig. 1. Information flows in multilevel security.

the whole flow of information, rather than the accesses of subjects to objects [36]. By imposing some information flow rules, it is possible to indifferently control direct and indirect leakages, as, in this perspective, they both become "unwanted information flows".

In the literature, there are many different security definitions reminiscent of the information flow idea, each based on some system model (see, e.g., [36, 64, 41, 48, 49, 62, 7]). In [24] we have compared and classified them, leading to our proposed notion of *Bisimulation Non Deducibility on Compositions* (*BNDC*, for short). We will present *BNDC* starting by the idea of Non Interference [36]. Through a running example and a comparison with other existing approaches, we will try to convince the reader that such a property can effectively detect unwanted information flows in systems, both direct and indirect.

We now describe the topics of the next sections. Section 2 presents the *Security Process Algebra* (SPA, for short) language. All the properties we will present and apply to the analysis of systems and protocols are based on such a language. SPA is an extension of CCS [50] – a language proposed to specify concurrent systems. The basic building blocks are the atomic activities, simply called *actions*; unlike CCS, in SPA actions belong to two different levels of confidentiality, thus allowing the specification of multilevel (actually, two-level) systems. As for CCS, the model used to describe the operational semantics of SPA is the *labelled transition system* model [43], where the states are the terms of the algebra. In order to express that certain states are indistinguishable for an external observer, semantic equivalences over terms/states are defined such that two terms are observationally indistinguishable iff they are equivalent. As explained below, the information flow security properties we introduce are all based on these notions of observable behaviours.

Section 3 is about such properties, that capture the existence of information flows among groups of users. We will see that these properties are all of the following algebraic form. Let E be an SPA process term, let X be a security

Finally, in Section 5 we give some concluding remarks and discuss some open problems.

2 SPA and Value-Passing

In this Section we present the language that will be used to specify and analyze security properties over concurrent systems. We first present the "pure" version of the language. Then we show how to extend it with value-passing. Finally, we present an example of value-passing agent specification. It will be our running example for the next sections.

2.1 The Language

The *Security Process Algebra* (SPA for short) [24, 26] is a slight extension of Milner's CCS [50], where the set of visible actions is partitioned into high level actions and low level ones in order to specify multilevel systems. [2]

SPA syntax is based on the same elements as CCS. In order to obtain a partition of the visible actions into two levels, we consider two sets Act_H and Act_L of high and low level actions which are closed with respect to function $\bar{\cdot}$ (i.e., $\overline{Act_H} = Act_H$, $\overline{Act_L} = Act_L$); moreover they form a covering of $\mathcal{L}$ and they are disjoint (i.e., $Act_H \cup Act_L = \mathcal{L}$, $Act_H \cap Act_L = \emptyset$). Let Act be the set $Act_H \cup Act_L \cup \{\tau\}$, where τ is a special unobservable, internal action. The syntax of SPA *agents* (or *processes*) is defined as follows:

$$E ::= \underline{0} \mid \mu.E \mid E + E \mid E|E \mid E\backslash\backslash L \mid E[f] \mid Z$$

where μ ranges over Act, $L \subseteq \mathcal{L}$ and $f : Act \to Act$ is such that $f(\bar{\alpha}) = \overline{f(\alpha)}$, $f(\tau) = \tau$. Moreover, for every constant Z there must be the corresponding definition: $Z \stackrel{\text{def}}{=} E$, and E must be *guarded* on constants. This means that the recursive substitution of all the non prefixed (i.e., not appearing in a context $\mu.E'$) constants in E with their definitions terminates after a finite number of steps. In other words, there exists a term obtainable by constant substitutions from E where all the possible initial actions are explicitly represented (through the prefix operator $\mu.E$). For instance, agent $A \stackrel{\text{def}}{=} B$ with $B \stackrel{\text{def}}{=} A$ is not guarded on constants. On the contrary, if B is defined as $a.A$, then B is guarded on constants. This condition will be useful when we will do automatic checks over SPA terms. As a matter of fact, it basically avoids infinite constant substitution loops.

Intuitively, we have that $\underline{0}$ is the empty process, which cannot do any action; $\mu.E$ can do an action μ and then behaves like E; $E_1 + E_2$ can alternatively

[2] Actually, only two-level systems can be specified; note that this is not a real limitation because it is always possible to deal with the multilevel case by grouping – in several ways – the various levels in two clusters.

Prefix	$\dfrac{-}{\mu.E \xrightarrow{\mu} E}$
Sum	$\dfrac{E_1 \xrightarrow{\mu} E_1'}{E_1 + E_2 \xrightarrow{\mu} E_1'} \quad \dfrac{E_2 \xrightarrow{\mu} E_2'}{E_1 + E_2 \xrightarrow{\mu} E_2'}$
Parallel	$\dfrac{E_1 \xrightarrow{\mu} E_1'}{E_1 \mid E_2 \xrightarrow{\mu} E_1' \mid E_2} \quad \dfrac{E_2 \xrightarrow{\mu} E_2'}{E_1 \mid E_2 \xrightarrow{\mu} E_1 \mid E_2'} \quad \dfrac{E_1 \xrightarrow{\alpha} E_1' \;\; E_2 \xrightarrow{\bar{\alpha}} E_2'}{E_1 \mid E_2 \xrightarrow{\tau} E_1' \mid E_2'}$
Restriction	$\dfrac{E \xrightarrow{\mu} E'}{E \setminus\setminus L \xrightarrow{\mu} E' \setminus\setminus L}$ if $\mu \notin L$
Relabelling	$\dfrac{E \xrightarrow{\mu} E'}{E[f] \xrightarrow{f(\mu)} E'[f]}$
Constant	$\dfrac{E \xrightarrow{\mu} E'}{A \xrightarrow{\mu} E'}$ if $A \stackrel{\text{def}}{=} E$

Table 1. The operational rules for SPA.

as for CCS by structural induction as the least relation generated by the axioms and inference rules reported in Table 1. The operational semantics for an agent E is the subpart of the SPA LTS reachable from the initial state E. We denote with $\mathcal{E}_{FS}$ the set of all the SPA agents with a finite LTS as operational semantics. Table 2 shows some simple examples of SPA terms with their corresponding LTSs.

Now we introduce the idea of observable behaviour: two systems should have the same semantics if and only if they cannot be distinguished by an external observer. To obtain this we define an equivalence relation over states/terms of the SPA LTS, equating two processes when they are indistinguishable. In this way the semantics of a term becomes an equivalence class of terms.

It is possible to define various equivalences of this kind, according to the different assumptions on the power of observers. We recall three of them. The first one is the classic definition of *trace equivalence*, according to which two agents are equivalent if they have the same execution traces. The second one discriminates agents also according to the nondeterministic structure of their LTSs. This equivalence is based on the concept of *bisimulation* [50]. The last one, introduced for the CSP language [39], is able to observe which actions are not executable after a certain trace (*failure* sets), thus detecting possible deadlocks.

Since we want to focus only on observable actions, we need a transition relation which does not take care of internal τ moves. This can be defined as follows:

Definition 1. *The expression* $E \stackrel{\alpha}{\Longrightarrow} E'$ *is a shorthand for* $E(\stackrel{\tau}{\rightarrow})^* E_1 \stackrel{\alpha}{\rightarrow} E_2(\stackrel{\tau}{\rightarrow})^* E'$, *where* $(\stackrel{\tau}{\rightarrow})^*$ *denotes a (possibly empty) sequence of* τ *labelled transitions. Let* $\gamma = \alpha_1 \ldots \alpha_n \in \mathcal{L}^*$ *be a sequence of actions; then* $E \stackrel{\gamma}{\Longrightarrow} E'$ *if and only if there exist* $E_1, E_2, \ldots, E_{n-1} \in \mathcal{E}$ *such that* $E \stackrel{\alpha_1}{\Longrightarrow} E_1 \stackrel{\alpha_2}{\Longrightarrow} \cdots \stackrel{\alpha_{n-1}}{\Longrightarrow} E_{n-1} \stackrel{\alpha_n}{\Longrightarrow} E'$. *For the empty sequence* $\langle\rangle$ *we have that* $E \stackrel{\langle\rangle}{\Longrightarrow} E'$ *stands for* $E(\stackrel{\tau}{\rightarrow})^* E'$. *We say that* E' *is* reachable *from* E *when* $\exists \gamma : E \stackrel{\gamma}{\Longrightarrow} E'$ *and we write* $E \Longrightarrow E'$. ■

Trace Equivalence We define trace equivalence as follows:

Definition 2. *For any* $E \in \mathcal{E}$ *the set* $T(E)$ *of* traces associated with E *is defined as follows:* $T(E) = \{\gamma \in \mathcal{L}^* \mid \exists E' : E \stackrel{\gamma}{\Longrightarrow} E'\}$. E *and* F *are* trace equivalent *(notation* $E \approx_T F$*) if and only if* $T(E) = T(F)$. ■

Observational Equivalence Bisimulation is based on an idea of mutual step-by-step simulation, i.e., when E executes a certain action moving to E', F must be able to simulate this single step by executing the same action and moving to an agent F' which is again bisimilar to E' (this is because it must be able to simulate every successive step of E'), and vice-versa.

A weak bisimulation is a bisimulation which does not care about internal τ actions. So, when F simulates an action of E, it can also execute some τ actions before or after that action.

In the following, $E \stackrel{\hat{\mu}}{\Longrightarrow} E'$ stands for $E \stackrel{\mu}{\Longrightarrow} E'$ if $\mu \in \mathcal{L}$, and for $E\ (\stackrel{\tau}{\rightarrow})^*\ E'$ if $\mu = \tau$ (note that $(\stackrel{\tau}{\rightarrow})^*$ means "zero or more τ labelled transitions" while $\stackrel{\tau}{\Longrightarrow}$ requires at least one τ labelled transition).

Definition 3. *A relation* $R \subseteq \mathcal{E} \times \mathcal{E}$ *is a* weak bisimulation *if* $(E, F) \in R$ *implies, for all* $\mu \in Act$,

- *whenever* $E \stackrel{\mu}{\longrightarrow} E'$, *then there exists* $F' \in \mathcal{E}$ *such that* $F \stackrel{\hat{\mu}}{\Longrightarrow} F'$ *and* $(E', F') \in R$;
- *conversely, whenever* $F \stackrel{\mu}{\longrightarrow} F'$, *then there exists* $E' \in \mathcal{E}$ *such that* $E \stackrel{\hat{\mu}}{\Longrightarrow} E'$ *and* $(E', F') \in R$.

Two SPA agents $E, F \in \mathcal{E}$ *are* observationally equivalent, *notation* $E \approx_B F$, *if there exists a weak bisimulation containing the pair* (E, F). ■

In [50] it is proved that $\approx_B$ is an equivalence relation. Moreover, it is easy to see that $E \approx_B F$ implies $E \approx_T F$; indeed, if $E \approx_B F$ then F must be able to simulate every sequence of visible actions executed by E, i.e., every trace of E; since the simulation corresponds to the execution of the actions interleaved with some τ's, then every trace of E is also a trace for F. Symmetrically, E must be able to simulate every sequence of F. So $E \approx_T F$.

In Figure 2 there is an example of two trace-equivalent systems which are not observationally equivalent. In fact both E and F can execute the three traces a, ab and ac. However, it is not possible for E to simulate step-by-step process F.

where $\approx_B S \approx_B$ *is the composition of binary relations, so that* $E' \approx_B S \approx_B F'$ *means that for some* E'' *and* F'' *we have* $E' \approx_B E''$, $(E'', F'') \in S$, $F' \approx_B F''$. ■

In [50] it is proven the following result:

Proposition 1. *If* S *is a bisimulation up to* $\approx_B$ *then* $S \subseteq \approx_B$.

Hence, to prove $P \approx_B Q$, we only have to find a bisimulation up to $\approx_B$ which contains (P, Q). This is one of the proof techniques we will often adopt in the following.

Failure/Testing Equivalence The failure semantics [9], introduced for the CSP language, is a refinement of the trace semantics where it is possible to observe which actions are not executable after a certain trace. In particular, a system is characterized by the so-called failures set, i.e., a set of pairs (s, X) where s is a trace and X is a set of actions. For each pair (s, X), the system must be able, by executing trace s, to reach a state where every action in X cannot be executed.[4] For instance, consider again agents E' and F' of Figure 3. As we said above, E' can stop after the execution of a and, consequently, E' can refuse to execute action b after the execution of a. So E' has the pair $(a, \{b\})$ in its failure set. System F' is always able to execute b after the execution of a. So F' does not have $(a, \{b\})$ in the failure set, hence it is not failure equivalent to E'. We deduce that also failure semantics is able to detect deadlocks. Moreover, also systems E and F of Figure 2 are not failure equivalent.

A different characterization of failure equivalence (called *testing equivalence*) has been given in [52]. It is based on the idea of tests. We can see a test T as any SPA process which can execute a particular *success* action $\omega \notin \mathcal{L}$. A test T is applied to a system E using the parallel composition operator $|$. A test T *may be satisfied* by system E if and only if system $(E|T) \setminus \mathcal{L}$ can execute ω. Note that in system $(E|T) \setminus \mathcal{L}$ we force the synchronization of E with test T.

Definition 5. E *may* T *if and only if* $(E|T) \setminus \mathcal{L} \stackrel{\omega}{\Longrightarrow} (E'|T') \setminus \mathcal{L}$ ■

A maximal computation of $(E|T) \setminus \mathcal{L}$ is a sequence $(E|T) \setminus \mathcal{L} = ET_0 \stackrel{\tau}{\rightarrow} ET_1 \stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow} ET_n \stackrel{\tau}{\rightarrow} \ldots$ which can be finite or infinite; if it is finite the last term must have no outgoing transitions. A test T *must be satisfied* by E if and only if every maximal computation of $(E|T) \setminus \mathcal{L}$ contains a state ET_i which can execute ω.

Definition 6. E *must* T *if and only if for all maximal computations of* $ET_0 = (E|T) \setminus \mathcal{L}, \exists i$ *such that* $ET_i \stackrel{\omega}{\rightarrow} ET_i'$ ■

Now we can define testing equivalence as follows:

Definition 7. *Two systems* E *and* F *are testing equivalent,* $E \approx_{test} F$, *if and only if*

[4] Indeed, there is a condition on the traces s in pairs (s, X). It is required that during any execution of s no infinite internal computation sequences are possible. We will analyze this aspect more in detail in the following.

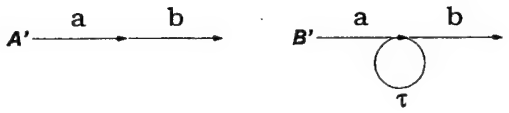


Fig. 5. Systems observationally equivalent but not testing equivalent.

aspect will be analyzed more in detail in the next chapters, when we will try to use $\approx_{test}$ in the specification of security properties.

We conclude this section presenting a class of finite state agents. This will be useful in the automatic verification of security properties. This class of agents consists of the so-called *nets of automata*:

$$p ::= \underline{0} \mid \mu.p \mid p+p \mid Z$$
$$E ::= p \mid E|E \mid E \setminus L \mid E \setminus_I L \mid E/L \mid E[f]$$

where for every constant Z there must be the corresponding definition $Z \stackrel{\text{def}}{=} p$ (with p guarded on constants). It is easy to prove that every agent of this form is finite state. However, this condition is not necessary, in the sense that other agents not belonging to the class of nets of automata are finite state as well. For instance, consider $B \stackrel{\text{def}}{=} a.\underline{0} + D \setminus \{i\}$ with $D \stackrel{\text{def}}{=} i.(o.\underline{0}|D)$. It can execute only an action a and then it stops, so it is clearly finite state. However note that it does not conform to the syntax of nets of automata since there is a parallel operator underneath a sum.

2.3 Value-Passing SPA

In this section we briefly present a value-passing extension of "pure" SPA (VSPA, for short). All the examples contained in this paper will use this value passing calculus, because it leads to more readable specifications than those written in pure SPA. Here we present a very simple example of a value-passing agent showing how it can be translated into a pure SPA agent. Then we define the VSPA syntax and we sketch the semantics by translating a generic VSPA agent into its corresponding SPA agent.

As an example, consider the following buffer cell [50]:

$$C \stackrel{\text{def}}{=} in(x).C'(x)$$
$$C'(x) \stackrel{\text{def}}{=} \overline{out}(x).C$$

where x is a variable that can assume values in $\mathbf{N}$ (we usually write $x \in \mathbf{N}$). C reads a natural number n through action in and stores it in variable x. Then this value is passed to agent C' which can give n as output through action out

We will use the notation $E\{a/b\}$ to represent agent E with all the occurrences of b substituted by a. We will also use $\mathcal{E}^+$ to denote the set of VSPA agents. For each agent $E \in \mathcal{E}^+$ without free variables, its translation $[\![E]\!]$ is given in Table 3 where $ari(a) = S_1 \ldots S_n$; $\widehat{L} = \{l_{v_1,\ldots,v_n} : l \in L, ari(l) = S_1 \ldots S_n, v_i \in$

$E \in \mathcal{E}^+$	$[\![E]\!] \in \mathcal{E}$
$\underline{0}$	$\underline{0}$
$a(x_1,\ldots,x_n).E$	$\sum_{i\in[1,n],v_i\in S_i} a_{v_1,\ldots,v_n}.[\![E\{v_1/x_1,\ldots v_n/x_n\}]\!]$
$\bar{a}(e_1,\ldots,e_n).E$	$\bar{a}_{\widehat{e_1},\ldots,\widehat{e_n}}.[\![E]\!]$
$\tau.E$	$\tau.[\![E]\!]$
$E_1 + E_2$	$[\![E_1]\!] + [\![E_2]\!]$
$E_1 \mid E_2$	$[\![E_1]\!] \mid [\![E_2]\!]$
$E \setminus L$	$[\![E]\!] \setminus \widehat{L}$
$E \setminus_I L$	$[\![E]\!] \setminus_I \widehat{L}$
E/L	$[\![E]\!]/\widehat{L}$
$E[f]$	$[\![E]\!][\widehat{f}]$
$A(e_1,\ldots,e_n)$	$A_{\widehat{e_1},\ldots,\widehat{e_n}}$
if b **then** E	$\begin{cases} [\![E]\!] \text{ if } \widehat{b} = True \\ \underline{0} \quad \text{otherwise} \end{cases}$

Table 3. Translation of VSPA to SPA.

$S_i, \forall i \in [1,n]\}$ is the set of the translations of actions in L and $\widehat{f}(l_{v_1,\ldots,v_n}) = f(l)_{v_1,\ldots,v_n}$ is the translation of relabelling function f. Furthermore, the single definition $A(x_1,\ldots,x_m) \stackrel{\text{def}}{=} E$ with $ari(A) = S_1 \ldots S_m$, is translated to the set of equations:

$$\{A_{v_1,\ldots,v_m} = [\![E\{v_1/x_1,\ldots,v_m/x_m\}]\!]; v_i \in S_i \forall i \in [1,m]\}$$

Note that we do not partition the set of actions into two levels; we directly refer to the partition in the pure calculus. In this way it is possible for a certain action in VSPA to correspond, in the translation, to actions at different levels in SPA. This can be useful if we want a parameter representing the level of a certain action. As an example consider an action $access\_r(l,x)$ with $l \in \{high, low\}$ and $x \in [1,n]$, representing a read request from a user at level l to an object x; we can assign the high level to the actions with $l = high$ and the low level to the others in this way: $access\_r(high,x) \in Act_H$ and $access\_r(low,x) \in Act_L$ for all $x \in [1,n]$. [6]

[6] Note that $access\_r(high,x)$ stands for $access\_r_{high,x}$, with $x \in [1,n]$. Indeed, for the sake of readability, we often write $c(v)$ instead of its translation c_v.

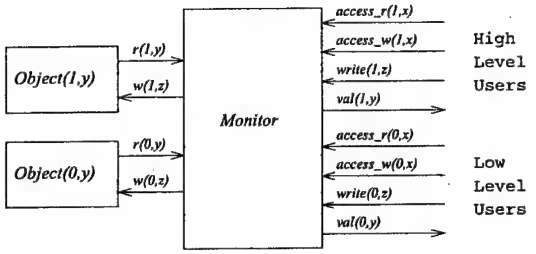


Fig. 6. The Access Monitor for Example 1.

only from the low one. Users interact with the monitor through the following access actions: $access\_r(l,x), access\_w(l,x), write(l,z)$ where l is the user level ($l = 0$ low, $l = 1$ high), x is the object ($x = 0$ low, $x = 1$ high) and z is the binary value to be written.

As an example, consider $access\_r(0,1)$ which represents a low level user ($l = 0$) read request from the high level object ($x = 1$), and $access\_w(1,0)$ followed by $write(1,0)$ which represents a high level user ($l = 1$) write request of value 0 ($z = 0$) on the low object ($x = 0$). Read results are returned to users through the output actions $val(l,y)$. This can be also an error in case of a read-up request. Note that if a high level user tries to write on the low object – through $access\_w(1,0)$ followed by $write(1,z)$ – such a request is not executed and no error message is returned.

In order to understand how the system works, let us consider the following transitions sequence representing the writing of value 1 in the low level object, performed by the low level user:

$$
\begin{aligned}
&(Monitor \mid Object(1,0) \mid Object(0,0)) \setminus L \\
&\xrightarrow{access\_w(0,0)} (write(0,z).\overline{w}(0,z).Monitor \mid Object(1,0) \mid Object(0,0)) \setminus L \\
&\xrightarrow{write(0,1)} (\overline{w}(0,1).Monitor \mid Object(1,0) \mid Object(0,0)) \setminus L \\
&\xrightarrow{\tau} (Monitor \mid Object(1,0) \mid Object(0,1)) \setminus L
\end{aligned}
$$

The trace corresponding to this sequence of transitions is

$$access\_w(0,0).write(0,1)$$

and so we can write:

$$
\begin{aligned}
&(Monitor \mid Object(1,0) \mid Object(0,0)) \setminus L \\
&\xRightarrow{access\_w(0,0).write(0,1)} (Monitor \mid Object(1,0) \mid Object(0,1)) \setminus L
\end{aligned}
$$

Note that, after the execution of the trace, the low level object contains value 1.

interfere with the low level if the effects of high level inputs are not visible by a low level user. This idea can be rephrased on the LTS model as follows. We consider every trace γ of the system containing high level inputs. Then, we look if there exists another trace γ' with the same subsequence of low level actions and without high inputs. A low level user, which can only observe low level actions, is not able to distinguish γ and γ'. As both γ and γ' are legal traces, we can conclude that the possible execution of the high level inputs in γ has no effect on the low level view of the system.

As for NI, we can define this property by using some functions which manipulates sequences of actions. In particular, it is sufficient to consider the function $low : \mathcal{L}^* \longrightarrow Act_L^*$ which takes a trace γ and removes all the high level actions from it, i.e., returns the low level subsequence of γ. Moreover we use the function $highinput : \mathcal{L}^* \longrightarrow (Act_H \cap I)^*$ which extracts from a trace the subsequence composed of all the high level inputs.

Definition 8. *(NNI: Non-deterministic Non Interference)*
$E \in NNI$ if and only if $\forall \gamma \in T(E), \exists \delta \in T(E)$ such that

(i) $low(\gamma) = low(\delta)$
(ii) $highinput(\delta) = \langle\rangle$

where $\langle\rangle$ denotes the empty sequence. ■

This may be expressed algebraically as:

Proposition 2. $E \in NNI \iff (E \setminus_I Act_H)/Act_H \approx_T E/Act_H$.

PROOF. ($\Rightarrow$) It is enough to show that if E is NNI then $T(E/Act_H) \subseteq T((E \setminus_I Act_H)/Act_H)$, because the opposite inclusion simply derives from $T(E \setminus_I Act_H) \subseteq T(E)$. Let $\gamma \in T(E/Act_H)$; then, by definition of / operator, $\exists \gamma' \in T(E)$ such that $low(\gamma') = \gamma$. Since $E \in NNI$ then $\exists \delta \in T(E)$ such that $low(\gamma') = low(\delta)$ and $highinput(\delta) = \langle\rangle$. Hence $\delta \in T(E \setminus_I Act_H)$ and $\delta' = low(\delta) \in T((E \setminus_I Act_H)/Act_H)$. Since $\gamma = low(\gamma') = low(\delta) = \delta'$, then $\gamma = \delta'$ and thus $\gamma \in T((E \setminus_I Act_H)/Act_H)$.
($\Leftarrow$) Let $\gamma \in T(E)$. Then $\exists \delta \in T(E \setminus_I Act_H)$ such that $low(\gamma) = low(\delta)$. Since $\delta \in T(E \setminus_I Act_H)$ then $highinput(\delta) = \langle\rangle$ and $\delta \in T(E)$. ■

As a matter of fact, in E/Act_H all the high level actions are hidden, hence giving the low level view of the system; $E \setminus_I Act_H$ instead prevents traces from containing high level inputs. So, if the two terms are equivalent, then for every trace with high level inputs we can find another trace without such actions but with the same subsequence of low level actions.

In the following we will consider this algebraic characterization as the definition of *NNI*. Indeed, all the other properties we present below in this section are defined using this compact algebraic style. An interesting advantage of this style is that it reduces the check of a security property to the "standard" and well studied problem of checking the semantic equivalence of two LTSs.

It is possible to prove that $Access\_Monitor\_1$ of Example 1 is *NNI*. In fact, the next example shows that *NNI* is able to detect whether the multilevel access control rules are implemented correctly in the monitor.

is missing then there will be a particular execution where some classified information is disclosed to low level users (otherwise such a rule would be useless). This will certainly modify the low level view of the system making it not *NNI* for sure.

However, the next example shows that *NNI* is not adequate to deal with synchronous communications and, consequently, it is too weak for SPA agents.

Example 3. Consider $Access\_Monitor\_1$, and suppose that we want to add a high level output signal which informs high level users about write operations of low level users in the high level object. This could be useful to know the integrity of high level information. We obtain the VSPA agent of Table 6 with the new $written\_up$ action and where $\forall i \in \{0,1\}$, $written\_up(i) \in Act_H$.

$$
\begin{array}{l}
Access\_Monitor\_3 \stackrel{\text{def}}{=} (Monitor\_3 \mid Object(1,0) \mid Object(0,0)) \setminus L \\
\quad Monitor\_3 \stackrel{\text{def}}{=} access\_r(l,x). \\
\qquad\qquad (\ \textbf{if } x \leq l \textbf{ then} \\
\qquad\qquad\quad r(x,y).\overline{val}(l,y).Monitor\_3 \\
\qquad\qquad\ \textbf{else} \\
\qquad\qquad\quad \overline{val}(l,err).Monitor\_3) \\
\qquad\qquad + \\
\qquad\qquad access\_w(l,x).write(l,z). \\
\qquad\qquad (\ \textbf{if } x = l \textbf{ then} \\
\qquad\qquad\quad \overline{w}(x,z).Monitor\_3 \\
\qquad\qquad\ \textbf{else} \\
\qquad\qquad\quad \textbf{if } x > l \textbf{ then} \\
\qquad\qquad\qquad \overline{w}(x,z).\overline{written\_up}(z).Monitor\_3 \\
\qquad\qquad\quad \textbf{else} \\
\qquad\qquad\qquad Monitor\_3)
\end{array}
$$

Table 6. The $Access\_Monitor\_3$ System.

It is possible to prove that $Access\_Monitor\_3$ is *NNI*. However, consider the following trace of $Access\_Monitor\_3$:

$$\gamma = access\_w(0,1).write(0,0).\overline{written\_up}(0).access\_w(0,1).write(0,0)$$

where a low level user writes two times value 0 into the high level object. If we purge γ of high level actions (i.e. of $written\_up$) we obtain the following sequence:

$$\gamma' = access\_w(0,1).write(0,0).access\_w(0,1).write(0,0)$$

$$
\begin{aligned}
Access\_Monitor\_4 &\stackrel{\text{def}}{=} (Monitor\_4 \mid Object(1,0) \mid Object(0,0)) \setminus L \\
Monitor\_4 &\stackrel{\text{def}}{=} access\_r(l,x). \\
&\quad (\ \textbf{if } x \leq l \textbf{ then} \\
&\qquad r(x,y).\overline{val}(l,y).Monitor\_4 \\
&\quad \textbf{else} \\
&\qquad \overline{val}(l,err).Monitor\_4) \\
&\quad + \\
&\quad access\_w(l,x).write(l,z). \\
&\quad (\ \textbf{if } x \geq l \textbf{ then} \\
&\qquad \overline{w}(x,z).Monitor\_4 \\
&\quad \textbf{else} \\
&\qquad Monitor\_4) \\
&\quad + \\
&\quad h\_stop.\underline{0}
\end{aligned}
$$

Table 7. The *Access_Monitor_4* System.

will know that *Access_Monitor_4* is not blocked and so that *h_stop* has not been executed yet. In section 3.4 we will show how the subtle information flow caused by a potential deadlock can be exploited in order to construct an information channel from high level to low level. ■

In order to detect this kind of flows, it is necessary to use some notion of equivalence which is able to detect deadlocks. Note that by simply changing the equivalence notion in the definition of *SNNI* we obtain a security property which inherits all the observation power of the new equivalence notion. So, for detecting deadlocks, one obvious possibility could be the failure/testing setting [9, 52], that has been designed for this purpose.

3.2 Detecting High Level Deadlocks Through Failure/Testing Equivalences

Consider the version of *SNNI* based on testing equivalence:

Definition 10. *(testing SNNI)* $E \in TSNNI \iff E/Act_H \approx_{test} E \setminus Act_H$ ■

We have that $Access\_Monitor\_4 \notin TSNNI$. In fact it is sufficient to consider the test $T \stackrel{\text{def}}{=} \overline{access\_r}(0,0).\omega.\underline{0}$ which is able to detect the deadlock introduced by action *h_stop*. In particular, we have that:

$$Access\_Monitor\_4 \setminus Act_H \ \ must\ T$$

which represents the backup timer and sends periodically a signal in order to obtain a backup. It is an abstraction of a clock, since in SPA it is not possible to handle time directly.

$$Backup\_timer \stackrel{\text{def}}{=} \overline{backup}.Backup\_timer$$

Then we slightly modify the $Monitor$ process by inserting two actions which suspend its execution until the backup is finished.

$$\begin{aligned} Monitor\_B \stackrel{\text{def}}{=} & \ldots \\ & \text{the same as in } Access\_Monitor\_4 \\ & \ldots \\ & + start\_backup.end\_backup.Monitor \end{aligned}$$

The backup process is enabled by the timer, then it stops the monitor, reads the values of variables, stores them into two additional objects ($Object(2, y)$ and $Object(3, y)$) and resumes the monitor:

$$\begin{aligned} Backup \stackrel{\text{def}}{=} & \ backup. \\ & \overline{start\_backup}. \\ & r(0, y).r(1, z). \\ & \overline{w}(2, y).\overline{w}(3, z). \\ & \overline{end\_backup}. \\ & Backup \end{aligned}$$

The access monitor with backup is given by the following system:

$$\begin{aligned} Access\_Monitor\_B \stackrel{\text{def}}{=} & (Monitor\_B \mid Backup\_timer \mid Backup \mid Object(0,0) \mid \\ & \mid Object(1,0) \mid Object(2,0) \mid Object(3,0)) \setminus L \end{aligned}$$

where $L = \{r, w, start\_backup, end\_backup, backup\}$. As a result, the backup procedure of the system is something internal, i.e., an external user can see nothing of the backup task. This makes the system divergent. In fact, if the variable values are unchanged, then the backup procedure is a τ loop that moves the system to the same state where it started the backup. For weak bisimulation this is not a problem and we can analyze this new system as well. In particular, we can check with the CoSeC tool (presented in Section 4) that $Access\_Monitor\_B$ is observationally equivalent to $Access\_Monitor\_4$. This is enough to prove (Theorem 5) that every security analysis made on $Access\_Monitor\_4$ is valid also for $Access\_Monitor\_B$. In particular, $Access\_Monitor\_B$ is not secure because of the potential high level deadlock we have explicitly added in $Access\_Monitor\_4$.

On the other hand, if we try to analyze this system with some testing equivalence based property, we have an inaccurate result. Indeed $Access\_Monitor\_B$, differently from $Access\_Monitor\_4$, is $TSNNI$. This happens because process

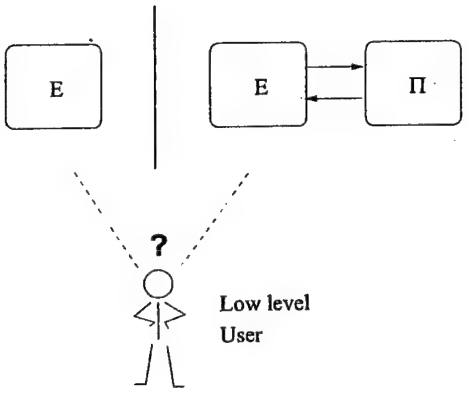


Fig. 8. BNDC intuition

PROOF. It immediately follows from the fact that $E \approx_B F$ implies $E \approx_T F$. The inclusions are proper because $E = \tau.l.0 + \tau.h.l.0$ is such that $E \in NNI, SNNI$ and $E \notin BNNI, BSNNI$. ■

Consider again *Access_Monitor_A* containing the *h_stop* event. It is neither *BSNNI* nor *BNNI*, as observational equivalence is able to detect deadlocks. In particular, $Access\_Monitor\_A/Act_H$ can move to $\underline{0}$ through an internal action τ, while $Access\_Monitor\_A \setminus Act_H$ is not able to reach (in zero or more τ steps) a state equivalent to $\underline{0}$.

Now we want to show that *BSNNI* and *BNNI* are still not able to detect some potential deadlocks due to high level activities. This will induce us to propose another property based on a different intuition. Let us consider *Access_Monitor_1*. We can prove that such a system is *BSNNI* as well as *BNNI*. However, the following two dangerous situations are possible: (*i*) a high level user makes a read request without accepting the corresponding output from the monitor (remember that communications in SPA are synchronous) and (*ii*) a high level user makes a write request and does not send the value to be written. In both cases we have a deadlock due to a high level activity that *BNNI* and *BSNNI* are not able to reveal. To solve this problem, we are going to present a stronger property, called *Bisimulation-based Non Deducibility on Compositions* (*BNDC*, for short). It is simply based on the idea of checking the system against all high level potential interactions. A system E is *BNDC* if for every high level process Π a low level user cannot distinguish E from $(E|\Pi) \setminus Act_H$. In other words, a system E is *BNDC* if what a low level user sees of the system is not modified by composing any high level process Π to E (see Figure 8).

Definition 12. *E is BNDC iff* $\forall \Pi \in \mathcal{E}_H$, $E/Act_H \approx_B (E \mid \Pi) \setminus Act_H$. ■

$$
\begin{aligned}
Access\_Monitor\_5 &\stackrel{\text{def}}{=} (AM \mid Interf) \setminus N \\
AM &\stackrel{\text{def}}{=} (Monitor\_5 \mid Object(1,0) \mid Object(0,0)) \setminus L \\
Monitor\_5 &\stackrel{\text{def}}{=} access\_r(l,x). \\
&\quad (\ \textbf{if } x \leq l \textbf{ then} \\
&\qquad r(x,y).\overline{val}(l,y).Monitor\_5 \\
&\quad \textbf{else} \\
&\qquad \overline{val}(l,err).Monitor\_5) \\
&\quad + \\
&\quad access\_w(l,x,z). \\
&\quad (\ \textbf{if } x \geq l \textbf{ then} \\
&\qquad \overline{w}(x,z).Monitor\_5 \\
&\quad \textbf{else} \\
&\qquad Monitor\_5) \\
Interf &\stackrel{\text{def}}{=} Interf(0) \mid Interf(1) \\
Interf(l) &\stackrel{\text{def}}{=} a\_r(l,x).\overline{access\_r}(l,x).val(l,k).\overline{put}(l,k).Interf(l) \\
&\quad + \\
&\quad a\_w(l,x,z).\overline{access\_w}(l,x,z).Interf(l)
\end{aligned}
$$

Table 8. The *Access_Monitor_5* System.

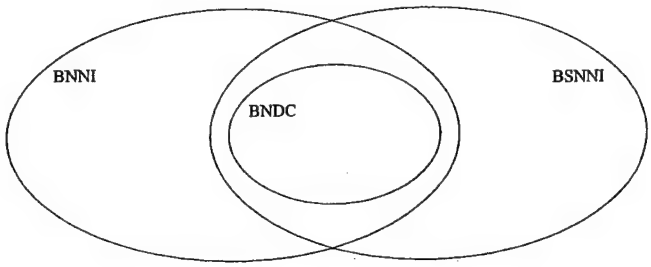


Fig. 10. The inclusion diagram for bisimulation-based properties

Figure 10 summarizes the relations among the bisimulation-based properties presented so far.

It is now interesting to study the trace equivalence version of *BNDC* called *NDC*. Indeed it could improve the *SNNI* property which is still our better proposal for the trace equivalence setting (no detection of deadlocks). Surprisingly, we find out that such property is exactly equal to *SNNI*.

Theorem 2. $NDC = SNNI$.

PROOF. We first prove that if $E \in NDC$ then $E \in SNNI$. By hypothesis, $E/Act_H \approx_T (E|0) \setminus Act_H$ for the specific $\Pi = 0$. Since $(E|0) \setminus Act_H \approx_T E \setminus Act_H$ then we have $E/Act_H \approx_T E \setminus Act_H$.

Now we want to prove that if $E \in SNNI$ then $E \in NDC$. By hypothesis, $E/Act_H \approx_T E \setminus Act_H$. Since $T(E \setminus Act_H) \subseteq T((E|\Pi) \setminus Act_H)$ then we have $T(E/Act_H) \subseteq T((E|\Pi) \setminus Act_H)$. Observe also that the reverse inclusion holds, in fact, if E and Π synchronize on a certain high action, then E/Act_H can always "hide" it. Hence $E/Act_H \approx_T ((E|\Pi) \setminus H)/Act_H$. ■

This, in a sense, confirms that the intuition behind *SNNI* (and so NI) is good at least in a trace equivalence model. Indeed, in such a model the fact that we check the system against every high level process is useless. It is sufficient to statically check if the hiding of high level actions corresponds to the restriction of such actions. This points out a critical point: *BNDC* is difficult to use in practice, because of the universal quantification on high level processes. It would be desirable to have an alternative formulation of *BNDC* which avoids universal quantification, exploiting local information only as for the trace-equivalence case; even if Martinelli has shown that *BNDC* is decidable over finite state processes [47], a solution to this problem is still to be found. In the same work, Martinelli also shows a negative fact regarding the verification of *BNDC*: it is not compositional, i.e., if two systems are *BNDC* their composition may be not so. This does not permit us to reduce the *BNDC*-security of a big system to the *BNDC*-security of its simpler subsystems and forces us to always prove *BNDC* over the whole system.

$(E''|F'')/Act_H$ which is simulated by $E'/Act_H|F'/Act_H \xrightarrow{\tau.\tau} E''/Act_H|F''/Act_H$. ■

Now we can prove the Theorem.

(i) Consider the relation $((E'|F') \setminus Act_H, E' \setminus Act_H|F' \setminus Act_H) \in R$ for all E', F' such that $E \Rightarrow E'$ and $F \Rightarrow F'$. If we prove that R is a weak bisimulation up to $\approx_B$ then, by hypothesis and Lemma 1, we obtain the thesis. We consider the only non trivial case: $(E'|F') \setminus Act_H \xrightarrow{\tau} (E''|F'') \setminus Act_H$ with $E' \xrightarrow{h} E''$ and $F' \xrightarrow{\bar{h}} F''$. Since $E'/Act_H \xrightarrow{\tau} E''/Act_H$ and, by hypothesis, $E' \in BSNNI$, we have that $\exists E'''$ such that $E' \setminus Act_H \stackrel{\hat{\tau}}{\Longrightarrow} E''' \setminus Act_H$ and $E''/Act_H \approx_B E''' \setminus Act_H$; finally, by hypothesis, $E'' \in BSNNI$, hence we obtain $E''' \setminus Act_H \approx_B E'' \setminus Act_H$. Repeating the same procedure for F' we have $\exists E''', F'''$ such that $E' \setminus Act_H|F' \setminus Act_H \stackrel{\hat{\tau}}{\Longrightarrow} E''' \setminus Act_H|F''' \setminus Act_H \approx_B E'' \setminus Act_H|F'' \setminus Act_H$. Since $((E''|F'') \setminus Act_H, E'' \setminus Act_H|\ F'' \setminus Act_H) \in R$, then R is a bisimulation up to $\approx_B$.

(ii) Consider the following relation $((E'/Act_H) \setminus L, (E' \setminus L)/Act_H) \in R$, for all E' such that $E \Rightarrow E'$ and for all $L \subseteq \mathcal{L}$. If we prove that R is a bisimulation up to $\approx_B$ then, by applying hypothesis and observing that $(E' \setminus L) \setminus Act_H \approx_B (E' \setminus Act_H) \setminus L$, we obtain the thesis. The only non trivial case is $(E'/Act_H) \setminus L \xrightarrow{\tau} (E''/Act_H) \setminus L$ with $E' \xrightarrow{h} E''$ and $h \in Act_H$. By hypothesis, $E' \in BSNNI$ hence we have that $(E'/Act_H) \setminus L \approx_B (E' \setminus Act_H) \setminus L$ and so $\exists E'''$ such that $(E' \setminus Act_H) \setminus L \stackrel{\hat{\tau}}{\Longrightarrow} (E''' \setminus Act_H) \setminus L \approx_B (E'''/Act_H) \setminus L$ and $(E''' \setminus Act_H) \setminus L \approx_B (E''/Act_H) \setminus L$. Obviously, we also have that $(E' \setminus L) \setminus Act_H \stackrel{\hat{\tau}}{\Longrightarrow} (E''' \setminus L) \setminus Act_H$ and so $(E' \setminus L)/Act_H \stackrel{\hat{\tau}}{\Longrightarrow} (E''' \setminus L)/Act_H$. We briefly summarize the proof: we had the synchronization $(E'/Act_H) \setminus L \xrightarrow{\tau} (E''/Act_H) \setminus L$ and we proved that there exists E''' such that $(E' \setminus L)/Act_H \stackrel{\hat{\tau}}{\Longrightarrow} (E''' \setminus L)/Act_H$ and $(E'''/Act_H) \setminus L \approx_B (E''/Act_H) \setminus L$. Since $((E'''/Act_H) \setminus L, (E''' \setminus L)/Act_H) \in R$ then R is a weak bisimulation up to $\approx_B$. ■

It is worthwhile noticing that *SBSNNI* was not the first sufficient condition proposed for *BNDC*. In [24] we introduced a property stronger than *SBSNNI*, but nevertheless quite intuitive, called *Strong BNDC* (*SBNDC*). This property just requires that before and after every high step, the system appears to be the same, from a low level perspective. More formally we have the following definition.

Definition 14. *(SBNDC: Strong BNDC)*
A system $E \in SBNDC$ if and only if $\forall E'$ reachable from E and $\forall E''$ such that $E' \xrightarrow{h} E''$ for $h \in Act_H$, then $E' \setminus Act_H \approx_B E'' \setminus Act_H$

We now prove that *SBNDC* is strictly stronger than *SBSNNI*. To this purpose, we need the following Lemma

Lemma 2. *$E \in BNDC \Leftrightarrow E \setminus Act_H \approx_B (E|\Pi) \setminus Act_H$ for all $\Pi \in \mathcal{E}_H$.*

PROOF. It follows immediately from Theorem 1.(ii) and Definition 12. ■

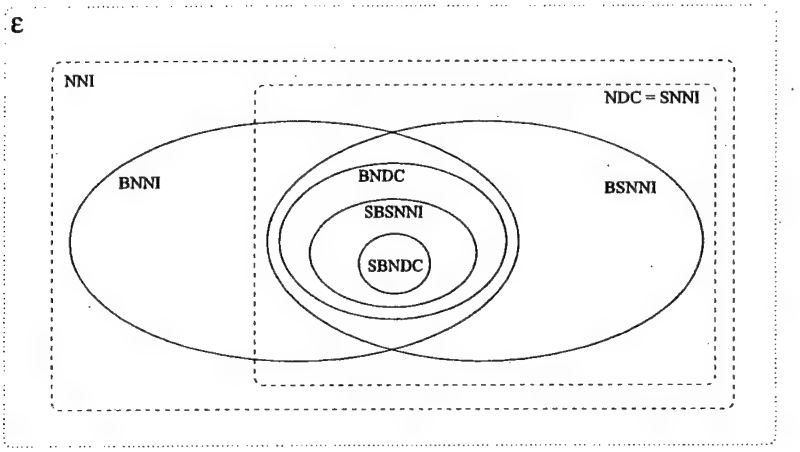


Fig. 11. The inclusion diagram for trace-based and bisimulation-based properties.

PROOF. It derives from the definition of the security properties, observing that trace and bisimulation equivalences are congruences with respect to $\_ \setminus L$ and $\_|\_$ operators of CCS [50]. It is possible to prove that they are also congruence with with respect to $\_ \setminus_I L$ and $\_/L$ operators of SPA. For trace equivalence the proof is trivial. In the following we prove the closure of weak bisimulation equivalence w.r.t. $\_ \setminus_I L$ and leave to the reader the other similar case of $\_/L$. Let $E \approx_B F$; we want to prove that $E \setminus_I L \approx_B F \setminus_I L$. Consider a bisimulation R such that $(E, F) \in R$; then define the relation P as follows: $(E' \setminus_I L, F' \setminus_I L) \in P$ if and only if $(E', F') \in R$. P is a bisimulation too, in fact if $E' \setminus_I L \xrightarrow{\mu} E'' \setminus_I L$ then $\mu \notin L \cap I$ and $E' \xrightarrow{\mu} E''$. So $\exists F''$ such that $F' \stackrel{\hat{\mu}}{\Longrightarrow} F''$; since $\mu \notin L \cap I$ then $F' \setminus_I L \stackrel{\hat{\mu}}{\Longrightarrow} F'' \setminus_I L$ (and vice versa for $F' \setminus_I L \xrightarrow{\mu} F'' \setminus_I L$). ■

3.4 Building Channels by Exploiting Deadlocks

In Example 6 we have seen that *Access_Monitor_1* is not *BNDC* because of potential high level deadlocks. We said that a deadlock due to high level activity is visible from low level users, hence it gives some information about high level actions, and cannot be allowed. However, one could doubt that a high level deadlock is really dangerous and, in particular, that it can be exploited to transmit information from high to low. We demonstrate that it is indeed the case by simply showing that it is possible to build a 1-bit channel from high to low level using systems which contain high level deadlocks. In particular we obtain a 1-bit channel with some initial noise (before the beginning of the transmission), using three processes with high level deadlocks composed with other secure systems.

$$
\begin{aligned}
Channel &\stackrel{\text{def}}{=} (Ch\_write(0) \mid Ch\_start(0) \mid AM(0) \mid AM(1) \mid AM(2) \mid \\
&\quad \mid R \mid T) \setminus N \\
AM(n) &\stackrel{\text{def}}{=} Access\_Monitor\_1[check(n)/access\_r(0,0), \\
&\quad block(n)/access\_w(1,1), \\
&\quad unblock(n)/write(1,0)] \setminus \{access\_r, access\_w, write, val\} \\
Ch\_write(x) &\stackrel{\text{def}}{=} send\_write.Ch\_write(1) \\
&\quad + \\
&\quad clear\_write.Ch\_write(0) \\
&\quad + \\
&\quad \textbf{if } x = 1 \textbf{ then} \\
&\quad \overline{up\_write}.Ch\_write(0) \\
Ch\_start(0) &\stackrel{\text{def}}{=} Ch\_write(0)[send\_start/send\_write, up\_start/up\_write, \\
&\quad clear\_start/clear\_write] \\
R &\stackrel{\text{def}}{=} \overline{send\_write}.R \\
&\quad + \\
&\quad \overline{check}(2).\overline{send\_start}. \\
&\quad (\overline{check}(0).\overline{output}(0).R \\
&\quad + \\
&\quad \overline{check}(1).\overline{output}(1).R) \\
T &\stackrel{\text{def}}{=} \overline{block}(2).\overline{clear\_write}.up\_write.\overline{block}(0).\overline{block}(1).T1 \\
T1 &\stackrel{\text{def}}{=} \overline{clear\_start}.\overline{unblock}(2).up\_start.\overline{block}(2).\overline{clear\_write}. \\
&\quad input(y).\overline{unblock}(y).up\_write.\overline{block}(y).T1
\end{aligned}
$$

Table 10. The *Channel* process which exploits deadlocks

3.5 Comparison with Related Work

The use of process algebras to formalize information flow security properties is not new. In [57] it is possible to find a definition of Non Interference given on CSP [39]. It looks like *SNNI* with some side conditions on acceptable low level actions. This definition is recalled in [4], where a comparison with another information flow property is reported.

More recent results based on the CSP model are contained in [56], where the authors introduce some information flow security properties based on the notion of *deterministic views* and show how to automatically verify them using the CSP model checker FDR [55].

The most interesting property is *lazy security* (*L-Sec*) which, however, requires the absence of non-determinism in the low view of the system (i.e., when hiding high actions through interleaving) and for this reason we think it could be too restrictive in a concurrent environment. For example, all the low non-deterministic systems – such as $E = l.\bar{l_1} + l.\bar{l_2}$ – are considered not secure. In this section we compare those properties with ours using a failure-equivalence version of *BNDC*, called *FNDC* (see also [18] for more details). The main result is that *BNDC* restricted to the class of low-deterministic and non-divergent processes is equal to *L-Sec*.

Here we give a definition of failure equivalence which does not correspond completely to the original one [39]. Indeed, it does not consider possible divergences but this is not a problem since our comparison will focus on the class of non-divergent processes. We prefer this definition because it is very simple and is implied by $\approx_B$.

We need some simple additional notations. We write $E \stackrel{\mu}{\not\Rightarrow}$ to indicate that $\nexists E'$ such that $E \stackrel{\mu}{\Longrightarrow} E'$ and $E \stackrel{K}{\not\Rightarrow}$ with $K \subseteq \mathcal{L}$ stands for $\forall \mu \in K, E \stackrel{\mu}{\not\Rightarrow}$.

Definition 15. *If $\gamma \in T(E)$ and if, after executing γ, E can refuse all the actions in set $X \subseteq \mathcal{L}$, then we say that the pair (γ, X) is a failure of the process E. Formally we have that:*

$$failures(E) \stackrel{\text{def}}{=} \{(\gamma, X) \subseteq \mathcal{L}^* \times \mathbb{P}(\mathcal{L}) \mid \exists E' \text{ such that } E \stackrel{\gamma}{\Longrightarrow} E' \text{ and } E' \stackrel{X}{\not\Rightarrow}\}$$

When $failures(E) = failures(F)$ we write $E \approx_F F$ (failure equivalence). ■

We identify a process E with its failure set. So if $(\gamma, X) \in failures(E)$ we write $(\gamma, X) \in E$. Note that $\gamma \in T(E)$ if and only if $(\gamma, \emptyset) \in E$. So $E \approx_F F$ implies $E \approx_T F$.

We also have that $E \approx_B F$ implies $E \approx_F F$:

Proposition 8. *$E \approx_B F$ implies $E \approx_F F$.*

PROOF. Consider $E \approx_B F$ and $(\gamma, X) \in E$. We want to prove that $(\gamma, X) \in F$. Since $(\gamma, X) \in E$ we have that $\exists E'$ such that $E \stackrel{\gamma}{\Longrightarrow} E'$ and $E' \stackrel{X}{\not\Rightarrow}$. By definition

A, with $\mathcal{L}(E)/A$ as inverse (the same holds for $B/\mathcal{L}(F)$ and $\mathcal{L}(F)/B$). This expression means that the actions in E and F are first relabelled using the two disjoint sets A and B, then interleaved (no communication is possible) and finally renamed to their original labels.

Recall that a process is *divergent* if it can execute an infinite sequence of internal actions τ. As an example consider the agent $A \stackrel{\text{def}}{=} \tau.A + b.\underline{0}$ which can execute an arbitrary number of τ actions. We define *Nondiv* as the set of all the non-divergent processes.

We can now present the *lazy security* property [56]. This property implies that the obscuring of high level actions by interleaving does not introduce any non-determinism. The obscuring of high level actions of process E by interleaving is obtained considering process $E|||RUN_H$ where $RUN_H \stackrel{\text{def}}{=} \sum_{h \in Act_H} h.RUN_H$. In such a process an outside observer is not able to tell if a certain high level action comes from E or from RUN_H.

L-Sec also requires that $E|||RUN_H$ is non-divergent. [9] This is equivalent to requiring that E is non-divergent, because RUN_H is non-divergent and the $|||$ operator does not allow synchronizations (which could generate new τ actions).

Definition 17. $E \in$ *L-Sec* $\Leftrightarrow E|||RUN_H \in Det \cap Nondiv$. ■

In the following we want to show that *L-Sec* can only analyze systems which are *low-deterministic*, i.e., where after any low level trace γ no low level action l can be both accepted and refused. The low-determinism requirement is not strictly necessary to avoid information flows from high to low level. So, in some cases, *L-Sec* is too strong. As an example consider the following non-deterministic system without high level actions: $E \stackrel{\text{def}}{=} l.l'.\underline{0} + l.l''.\underline{0}$. It is obviously secure but it is not low-deterministic and so it is not *L-Sec*. Formally we have that:

Definition 18. *E is low-deterministic ($E \in$ Lowdet) iff* $E \setminus Act_H \in Det$. ■

The following holds:

Theorem 6. *L-Sec* $\subseteq$ *Lowdet.*

PROOF. Let $E \in$ *L-Sec*. Consider a trace γa of $E \setminus Act_H$ and suppose that $(\gamma, \{a\}) \in E \setminus Act_H$. So there exists E' such that $E \setminus Act_H \stackrel{\gamma}{\Longrightarrow} E' \setminus Act_H$ and such that $E' \setminus Act_H \stackrel{a}{\not\rightarrow}$. Since RUN_H cannot execute the low level action a then we have that $E'|||RUN_H \stackrel{a}{\not\rightarrow}$ and so $(\gamma, \{a\}) \in E|||RUN_H$ because $E|||RUN_H \stackrel{\gamma}{\Longrightarrow} E'|||RUN_H$. Since γa is a trace for $E \setminus Act_H$ then it is also a trace for $E|||RUN_H$ and we obtain that $E|||RUN_H$ is not deterministic, contradicting the hypothesis. So $(\gamma, \{a\}) \notin E \setminus Act_H$ and $E \in$ *Lowdet*. ■

[9] Note that in [56] the non-divergence requirement is inside the deterministic one. This is because the authors use the failure-divergence semantics [10]. In this work we use the failure equivalence which does not deal with divergences. So, in order to obtain exactly the *L-Sec* property, we require the non-divergence condition explicitly.

$traces(E'/Act_H)$. Since we have that $E \setminus Act_H \stackrel{\gamma}{\Longrightarrow} E' \setminus Act_H$ then $(E|\Pi) \setminus Act_H \stackrel{\gamma}{\Longrightarrow} (E'|\Pi) \setminus Act_H$ and so $(\gamma, K) \in (E|\Pi) \setminus Act_H$.

The inclusion is strict because agent $E \stackrel{\text{def}}{=} l.h.l.\underline{0} + l.\underline{0} + l.l.\underline{0}$ is *FNDC* but not *SFSNNI*.

(*FNDC* $\subset$ *FSNNI*) It is sufficient to consider $\Pi = \underline{0}$. We have that $(E|\underline{0}) \setminus Act_H \approx_F E \setminus Act_H$ and so, since $(E|\underline{0}) \setminus Act_H \approx_F E/Act_H$ we have $E/Act_H \approx_F E \setminus Act_H$.

The inclusion is strict because agent $E \stackrel{\text{def}}{=} l.h.l.h'.l.\underline{0} + l.\underline{0} + l.l.l.\underline{0}$ is *FSNNI* but not *FNDC*. ■

Figure 14 summarizes the inclusions among the presented security properties. It can be drawn using the previous inclusion results and the following remarks: *BNDC* $\not\subseteq$ *SFSNNI*, in fact agent $l.h.l.\underline{0} + l.\underline{0} + l.l.\underline{0}$ is *BNDC* but not *SFSNNI*; we also have that *BSNNI* $\not\subseteq$ *FNDC* because of agent $h.l.h'.l.\underline{0} + l.l.\underline{0}$; finally *SFSNNI* $\not\subseteq$ *BSNNI* because of agent $h.l.(l'.\underline{0} + l''.\underline{0}) + l.l'.\underline{0} + l.l''.\underline{0}$.

The next theorem shows that under the low-determinism assumption the properties *SFSNNI* and *FNDC* collapse into the same one. We need the following Lemma.

Lemma 3. *If* $E, \tilde{E} \in Det$, $E \stackrel{\gamma}{\Longrightarrow} E'$, $\tilde{E} \stackrel{\gamma}{\Longrightarrow} \tilde{E}'$ *and* $E \approx_F \tilde{E}$ *then* $E' \approx_F \tilde{E}'$.

PROOF. We prove that if $(\delta, K) \in E'$ then $(\delta, K) \in \tilde{E}'$. Let $(\delta, K) \in E'$. Then $(\gamma\delta, K) \in E$ and by $E \approx_F \tilde{E}$ we obtain that $(\gamma\delta, K) \in \tilde{E}$. So $\exists \tilde{E}'', \tilde{E}'''$ such that $\tilde{E} \stackrel{\gamma}{\Longrightarrow} \tilde{E}'' \stackrel{\delta}{\Longrightarrow} \tilde{E}''' \stackrel{K}{\not\Rightarrow}$, hence $(\delta, K) \in \tilde{E}''$. Since $\tilde{E} \in Det$ then by Proposition 9 and hypothesis we have that $\tilde{E}'' \approx_F \tilde{E}'$ and so $(\delta, K) \in \tilde{E}'$. We can prove in the same way that if $(\delta, K) \in \tilde{E}'$ then $(\delta, K) \in E'$. So $E' \approx_F \tilde{E}'$ ■

Theorem 8. *FNDC* $\cap$ *Lowdet* $\subseteq$ *SFSNNI*.

PROOF. Since *FNDC* $\subset$ *FSNNI* and $E \in$ *FNDC*, we have that $E \setminus Act_H \approx_F E/Act_H$. By $E \in$ *Lowdet* we obtain $E/Act_H \in Det$. Now consider $E \stackrel{\gamma}{\Longrightarrow} E'$. We have to prove that $E'/Act_H \approx_F E' \setminus Act_H$. Let Π' be the high level process which executes exactly the complement of the high level projection of γ, i.e. the complement of the subsequence of γ composed by all the high level actions in γ. If γ' is the low level projection of γ we have that $(E|\Pi') \setminus Act_H \stackrel{\gamma'}{\Longrightarrow} (E'|\underline{0}) \setminus Act_H \approx_F E' \setminus Act_H$. Since $E \stackrel{\gamma}{\Longrightarrow} E'$ then $E/Act_H \stackrel{\gamma'}{\Longrightarrow} E'/Act_H$. By hypothesis we have that $(E|\Pi') \setminus Act_H \approx_F E/Act_H$. Since $E/Act_H \in Det$ then, by Lemma 3, we have that $E'/Act_H \approx_F (E'|\underline{0}) \setminus Act_H \approx_F E' \setminus Act_H$. ■

Corollary 2. *FNDC* $\cap$ *Lowdet* $=$ *SFSNNI* $\cap$ *Lowdet*.

PROOF. Trivial by Theorems 8 and 7. ■

Comparison We now show that under the low-determinism and the non-divergence assumption the *BNDC* property is equal to *L-Sec*. We start proving this result for *FNDC*.

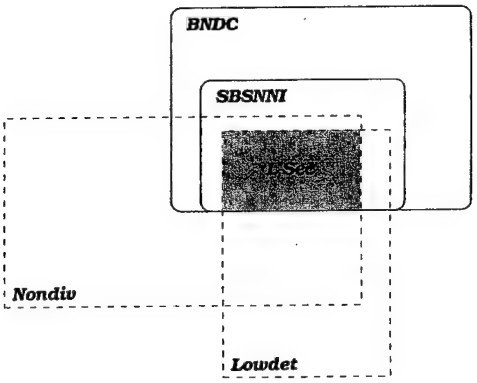


Fig. 15. Relations among properties.

contradict the fact that $E \in L\text{-}Sec$ and so $E''/Act_H \stackrel{a}{\not\Rightarrow}, \forall a \in K$. Hence $(\delta, K) \in E'/Act_H$. ■

Theorem 10. *SFSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv* $\subseteq$ *L-Sec.*

PROOF. Let $E \in SFSNNI \cap Lowdet \cap Nondiv$ and γa be a trace for process $E|||RUN_H$. We want to prove that $(\gamma, \{a\}) \notin E|||RUN_H$. It trivially holds if $a \in Act_H$ because in such a case it can always be executed by RUN_H. So let $a \in Act_L$. Suppose $E|||RUN_H \stackrel{\gamma}{\Longrightarrow} E'|||RUN_H \stackrel{a}{\not\Rightarrow}$ and consider the sequence γ' obtained removing all the high level actions from γ. Then $E/Act_H \stackrel{\gamma'}{\Longrightarrow} E'/Act_H$ and by hypothesis $E'/Act_H \approx_F E'\backslash Act_H$. Since $E'|||RUN_H \stackrel{a}{\not\Rightarrow}$ then $E'\backslash Act_H \stackrel{a}{\not\Rightarrow}$ and so $E'/Act_H \stackrel{a}{\not\Rightarrow}$ and $(\gamma', \{a\}) \in E/Act_H$. Since $E \in SFSNNI$ we obtain that $(\gamma', \{a\}) \in E\backslash Act_H$. Now γa is a trace for $E|||RUN_H$ and so $\gamma' a$ must be a trace for E/Act_H this means that $\gamma' a$ is also a trace for $E \setminus Act_H$. Since $E \in Lowdet$ then $E\backslash Act_H$ is deterministic. However we found that $\gamma' a$ is a trace for $E\backslash Act_H$ and $(\gamma', \{a\}) \in E\backslash Act_H$ obtaining a contradiction. So $E'|||RUN_H$ cannot refuse a and $(\gamma, \{a\}) \notin E|||RUN_H$. Hence $E|||RUN_H \in Det$ and since $E \in Nondiv$ we also have that $E|||RUN_H \in Nondiv$ ■

Corollary 3. *SFSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv* $=$ *L-Sec.*

PROOF. By Theorems 6 and 9 and by Definition 17 we find that L-Sec $\subseteq$ *SFSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv*. Finally by Theorem 10 we obtain the thesis. ■

Note that by Corollary 2 we also have that $FNDC \cap Lowdet \cap Nondiv = L\text{-}Sec$. Now we show that this result also holds for *SBSNNI* and *BNDC*. We first prove that for deterministic processes $\approx_F$ becomes equal to $\approx_B$.

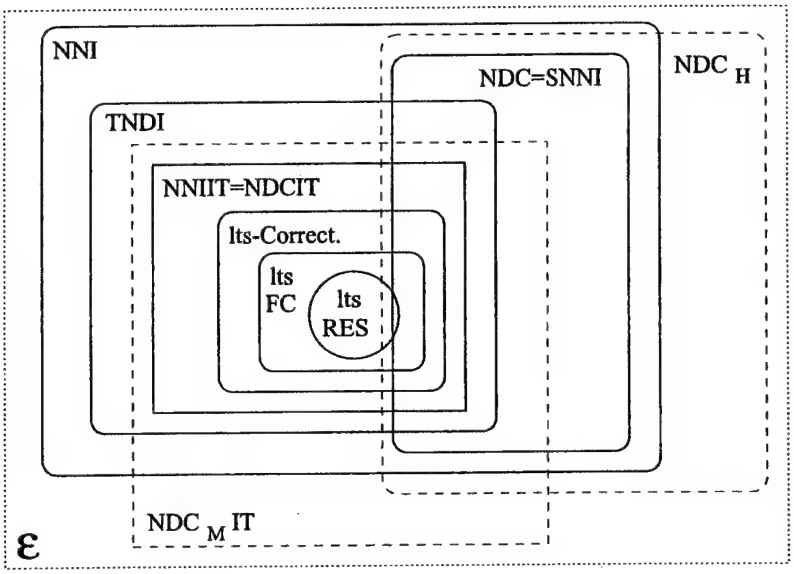


Fig. 16. The inclusion diagram for trace-based security properties.

A good reference about modelling NI in CSP can be found in this volume [58]. In such a paper, a new notion based on power bisimulation is also proposed. We intend to compare it with our bisimulation-based properties with the aim making our classification as much complete as possible.

3.6 Other Security Properties

In [24] we have compared our properties with a number of existing proposal. Here we just report the diagram (Figure 16) of the relations among such properties and we give the bibliographic references to them. In particular, TNDI derives from *Non Deducibility on Inputs* [62], lts-Correctability comes from *Correctability* [41], lts-FC is *Forward-Correctability* [41], lts-RES is a version of *Restrictiveness* [48]. Moreover NNIIT is *NNI* where we require that systems are *input total*. This means that in every state the system must be able to accept every possible input by the environment. The aim of this (quite restrictive) condition is to prevent users from deducing anything about the state of the system, even if they observe the inputs the system accepts. All the properties included in NNIIT requires this condition over input actions. NDCIT is *NDC* with input totality. Finally NDC_H and $\mathrm{NDC}_M\mathrm{IT}$ are parametric versions of *NDC* where the high level user can exploit only the actions in sets H and M to communicate with

	NNI	NDC SNNI	TNDI	lts-RES	lts-cor.	lts-FC	NDC_H	NNIIT NDCIT	NDC_MIT
NNI	=	8	2	2	2	2	9	10	10
NDC SNNI	$\subset$	=	2	2	2	2	$\subset$	10	10
TNDI	$\subset$	1	=	3	3	3	9	10	10
lts-RES	$\subset$	1	$\subset$	=	$\subset$	$\subset$	9	$\subset$	$\subset$
lts-cor.	$\subset$	1	$\subset$	6	=	6	9	$\subset$	$\subset$
lts-FC	$\subset$	1	$\subset$	5	$\subset$	=	9	$\subset$	$\subset$
NDC_H	7	7	7	7	7	7	=	10	10
NNIIT NDCIT	$\subset$	1	$\subset$	3	3	3	9	=	$\subset$
NDC_MIT	4	4	4	4	4	4	3	4	=

Let $T[x, y]$ be the table element contained in row x and column y. If $T[x, y] \in \{=, \subset\}$ then $xT[x, y]y$. If $T[x, y] = n$ then the agent n below is in x and is not in y. In the following Π_0 represents the Input-Total empty agent: $\Pi_0 = \sum_{i \in I} i.\Pi_0$.

1) $Z = \sum_{i \in I} i.Z + \bar{h}.Z'$ and $Z' = \sum_{i \in I} i.Z' + \bar{l}.\Pi_0$
2) $h.l.0 + \bar{l}.0 + l'.0$
3) $Z = \sum_{i \in I} i.Z + \bar{h}.(\Pi_0 + \bar{l}.\Pi_0)$
4) $Z = \sum_{i \in I} i.Z + h.(\Pi_0 + \bar{l}.\Pi_0)$ with $h \notin M \cup \bar{M}$
5) $Z = \sum_{i \in I} i.Z + \bar{l}.\Pi_0 + h.\Pi_0$
6) $Z = \sum_{i \in I} i.Z + \bar{h}.Z' + \bar{h}'.Z''$ and $Z' = \Pi_0 + h.Z' + \bar{l}.\Pi_0$ and $Z'' = \sum_{i \in I \setminus \{h\}} i.Z'' + h.\Pi_0 + \bar{l}.\Pi_0$
7) $h.l.0$ with $h \notin H \cup \bar{H}$
8) $\bar{h}.l.0$
9) $Z = \sum_{i \in I} i.Z + \bar{h}.Z'$ and $Z' = \sum_{i \in I} i.Z' + \bar{l}.\Pi_0$ with $\bar{h} \in H \cup \bar{H}$
10) 0

Table 11. The inclusion table for trace-based security properties.

In the analysis layer, CoSeC uses a routine of equivalence and one of minimization that belong to the analysis layer of CW. These are a slight modification of the algorithm by Kanellakis and Smolka [42] which finds a bisimulation between the roots of two finite state LTSs by partitioning their states. It is interesting to note that a simple modification of this algorithm can be used to obtain the minimization of a finite state LTS.

4.2 Checking the Information Flow Properties

Here we describe in details how the verification of Information Flow Properties proceeds. As we said before, we have properties which are in the following form:

$$E \text{ is } X\text{-secure if and only if } \mathcal{C}_X[E] \approx \mathcal{D}_X[E].$$

We have seen for *SNNI* that $\mathcal{C}_X[-] = - \setminus Act_H$ and $\mathcal{D}_X[-] = -/Act_H$. Hence, checking the X-security of E is reduced to the "standard" problem of checking semantic equivalence between two terms having E as a sub-term.

In the following we briefly explain how the system works in evaluating security predicates *NNI, SNNI, NDC, BNNI, BSNNI, SBSNNI*, and we discuss about their computational complexity. CoSeC computes the value of these predicates over finite state agents (i.e. agents possessing a finite state LTS), based on the definitions given in Section 3 that we report below in CoSeC syntax (for ease of parsing, in CoSeC the hiding and input restriction operators are represented by ! and ?, respectively.):

$$\begin{aligned}
E \in NNI &\Leftrightarrow E!Act_H \approx_T (E?Act_H)!Act_H \\
E \in SNNI \equiv NDC &\Leftrightarrow E!Act_H \approx_T E \setminus Act_H \\
E \in BNNI &\Leftrightarrow E!Act_H \approx_B (E?Act_H)!Act_H \\
E \in BSNNI &\Leftrightarrow E!Act_H \approx_B E \setminus Act_H \\
E \in SBSNNI &\Leftrightarrow E' \in BSNNI,\ \forall E' \textit{ reachable from } E
\end{aligned}$$

As for CW, the inner computation of the CoSeC follows three main phases.

Phase a) (the same for all predicates) CoSeC builds the RLTSs of the two agents of which it wants to compute the equivalence. For example in the case of NNI, CoSeC computes the transition graph for $(E?Act_H)!Act_H$ and $E!Act_H$. In this phase we do not have any particular problem with complexity, except for the intrinsic exponential explosion in the number of states of the RLTS due to parallel composition.

Phase b) (This is split into two depending on the semantics requested by the security predicate)

b1: (for predicates *NNI, SNNI, NDC*) The two RLTSs obtained in Phase a) are transformed into deterministic graphs following the classic subset construction (see e.g. [40]). This algorithm has exponential complexity since it is theoretically possible, in the deterministic graph, to have a node for every subset of nodes in the original graph. However, experience

```
Command: nni  A
true
```

CoSeC tells us that A is NNI secure. Now we can check if agent A is SNNI secure too:

```
Command: snni  A
false
```

So A is *NNI* secure but is not *SNNI* secure. If we want to know why such a system is not *SNNI* we can use the *debugging* version of the *SNNI*:

```
Command: d_snni  A
false
Agent A!ActH
can perform action sequence 'l
which agent A\ActH
cannot
```

The tool shows a (low level) trace which distinguishes processes A/Act_H and $A \setminus Act_H$. The trace is $\bar{l}$ which can be executed only by the first one. This can be useful to understand why a process is not secure. Finally the command **quit** causes an exit to the shell.

4.4 An Example: Checking the Access Monitor

In this Section we use CoSeC to automatically check all the versions of the access monitor discussed in Example 6. Since CoSeC works on SPA agents we have to translate all the VSPA specifications into SPA. Consider once more *Access_Monitor_1*. Table 12 reports the translation of *Access_Monitor_1* specification into the CoSeC syntax for SPA. [12] It has been used a new command **basi** which binds a set of actions to an identifier. Moreover, the \ character at the end of a line does not represent the restriction operator, but is the special character that permits to break in more lines the description of long agents and long action lists.

We can write to a file the contents of Table 12 and load it, in CoSeC, with command **if** <*filename*>. Now we can check that *Access_Monitor_1* satisfies all the security properties except *SBSNNI* using the following command lines:

```
Command: bnni  Access_Monitor_1
true
Command: bsnni  Access_Monitor_1
true
Command: sbsnni  Access_Monitor_1
false: ('val_h1.Monitor | Object_l1 | Object_h1) \ L
```

[12] In the translation, we use values $\{l, h\}$ in place of $\{0, 1\}$ for the levels of users and objects in order to make the SPA specification clearer. As an example $access\_r(1, 0)$ becomes `access_r_hl`.

Note that when CoSeC fails to verify *SBSNNI* on a process E, it gives as output an agent E' which is reachable from E and is not *BSNNI*.

So we have found that

$$Access\_Monitor\_1 \in BSNNI, BNNI$$

but

$$Access\_Monitor\_1 \notin SBSNNI$$

Since we have that $SBSNNI \subset BNDC \subset BSNNI, BNNI$, we cannot conclude whether $Access\_Monitor\_1$ is *BNDC* or not. However, using the output state E' of the *SBSNNI* verification, it is easy to find a high level process Π which can block the monitor. Indeed, in the state given as output by *SBSNNI*, the monitor is waiting for the high level action `'val_h1`; so, if we find a process Π which moves the system to such a state and does not execute the `val_h1` action, we will have a high level process able to block the monitor. It is sufficient to consider $\Pi = 'access\_r\_hh.0$. Agent $(Access\_Monitor\_1|\Pi) \setminus Act_H$ will be blocked immediately after the execution of the read request by Π, moving to the following deadlock state:

$$(('\texttt{val\_h0}.\texttt{Monitor} \mid \texttt{Object\_l0} \mid \texttt{Object\_h0}) \setminus \texttt{L} \mid \texttt{0}) \setminus \texttt{Act}_\texttt{H}$$

(this state differs from the one given as output by *SBSNNI* only for the values stored in objects). It is possible to verify that $Access\_Monitor\_1 \notin BNDC$ by checking that $(Access\_Monitor\_1|\Pi) \setminus Act_H \not\approx_B Access\_Monitor\_1/Act_H$ using the following command:

```
Command: bi  Pi   'access_r_hh.0
Command: eq
Agent: (Access_Monitor_1 | Pi) \ acth
Agent: Access_Monitor_1 ! acth
false
```

As we said in Example 6, such a deadlock is caused by synchronous communications in SPA. Moreover, using the CoSeC output again, we can find out that also the high level process $\Pi' = 'access\_w\_hl.0$ can block $Access\_Monitor\_1$, because it executes a write request and does not send the corresponding value. Hence, in Example 6 we proposed the modified system $Access\_Monitor\_5$ with an interface for each level and atomic actions for write request and value sending. We finally check that this version of the monitor is *SBSNNI*, hence *BNDC* too:

```
Command: sbsnni   Access_Monitor_5
true
```

4.5 State Explosion and Compositionality

In this section we show how the parallel composition operator can increase exponentially the number of states of the system, and then how it can slow down

- $E, E' \in P \Longrightarrow E|E' \in P$
- $E \in P, L \subseteq \mathcal{L} \Longrightarrow E \setminus L \in P$
- $E \in P, Z \stackrel{\text{def}}{=} E \Longrightarrow Z \in P$

and let A_P be a decision algorithm which checks if a certain agent $E \in \mathcal{E}_{FS}$ belongs to P; in other words, $A_P(E) = true$ if $E \in P$, $A_P(E) = false$ otherwise. Then we can define a compositional algorithm $A'_P(E)$ in the following way:

1) *if E is of the form $E' \setminus L$, then compute $A'_P(E')$; if $A'_P(E') = true$ then return true, else return the result of $A_P(E)$;*
2) *if E is of the form $E_1|E_2$, then compute $A'_P(E_1)$ and $A'_P(E_2)$; if $A'_P(E_1) = A'_P(E_2) = true$ then return true, else return the result of $A_P(E)$;*
3) *if E is a constant Z with $Z \stackrel{\text{def}}{=} E'$, then return the result of $A'_P(E')$;*
4) *if E is not in any of the three forms above, then return $A_P(E)$.* ■

The compositional algorithm $A'_P(E)$ works as the given algorithm $A_P(E)$ when the outermost operator of E is neither the restriction operator, nor the parallel one, nor a constant definition. Otherwise, it applies componentwise to the arguments of the outermost operator; if the property does not hold for them, we cannot conclude that the whole system is not secure, and we need to check it with the given algorithm.

Note that the compositional algorithm exploits the assumption that property P is closed with respect to restriction and uses this in step 1. This could seem of little practical use, as the dimension of the state space for, let say, E is often bigger than that of $E \setminus L$. However, parallel composition is often used in the form $(A|B) \setminus L$ in order to force some synchronizations, and so if we want to check P over A and B separately, we must be granted that P is preserved by both parallel and restriction operators.

To obtain the result for $A'_P(F)$, we essentially apply – in a syntax-driven way – the four rules above recursively, obtaining a proof tree having (the value of) $A'_P(F)$ as the root and the various (values of) $A_P(E)$'s on the leaves for the subterms E of F on which the induction cannot be applied anymore. The following theorem justifies the correctness of the compositional algorithm, by proving that the evaluation strategy terminates and gives the same result as the given algorithm $A_P(F)$.

Theorem 12. *Let $F \in \mathcal{E}_{FS}$. If the agent E' occurring in step 1 belongs to $\mathcal{E}_{FS}$ each time the algorithm A'_P executes that step, then $A'_P(F)$ terminates and $A_P(F) = A'_P(F)$.*

PROOF. First we want to prove that, in computing $A'_P(F)$, if the evaluation of the given algorithm A_P is required on an agent E, then E belongs to $\mathcal{E}_{FS}$. The proof is by induction on the proof tree for the evaluation of $A'_P(F)$. The base case is when F can be evaluated by step 4; as – by hypothesis – agent F is finite state, the thesis follows trivially. Instead, if F is of the form $E' \setminus L$, then – by the premise of this theorem – $E' \in \mathcal{E}_{FS}$, and the inductive hypothesis can be applied. In step 2, as $F = E_1|E_2$, we have that $E_1, E_2 \in \mathcal{E}_{FS}$, and the inductive

```
Command: c_sbsnni  Access_Monitor_5
[\] Verifying AM | Interf
  [|] Verifying AM
    [\] Verifying Monitor_5 | Object_10 | Object_h0
      [|] Verifying Monitor_5
      [|] Failed!
    [\] Failed!
        Verifying directly (Monitor_5 | Object_10 | Object_h0)\L
  [|] Failed!
[\] Failed!
    Verifying directly (AM | Interf)\K
true
```

Table 14. Verification of *SBSNNI* on *Access_Monitor_5* with the compositional algorithm.

$$
\begin{aligned}
Access\_Monitor\_6 &\stackrel{\text{def}}{=} (AM\_6 \mid Interf\_6) \setminus N \\
AM\_6 &\stackrel{\text{def}}{=} ((Monitor\_5 \mid Object(1,0) \mid Object(0,0) \\
&\qquad \mid hBuf(empty)) \setminus L)[res(0,y)/val(0,y)] \\
hBuf(j) &\stackrel{\text{def}}{=} \overline{res}(1,j).hBuf(empty) + val(1,k).hBuf(k) \\
Interf\_6 &\stackrel{\text{def}}{=} Interf\_6(0) \mid Interf\_6(1) \\
Interf\_6(l) &\stackrel{\text{def}}{=} a\_r(l,x).\overline{access\_r}(l,x).Interf\_6\_reply(l) \\
&\qquad + \\
&\qquad a\_w(l,x,z).\overline{access\_w}(l,x,z).Interf\_6(l) \\
Interf\_6\_reply(l) &\stackrel{\text{def}}{=} res(l,y). \\
&\qquad (\ \textbf{if } y = empty \textbf{ then} \\
&\qquad\quad Interf\_6\_reply(l) \\
&\qquad\ \textbf{else} \\
&\qquad\quad \overline{put}(l,y).Interf\_6(l))
\end{aligned}
$$

Table 15. The *Access_Monitor_6*.

In fact, suppose we want to add other objects to *Access_Monitor_6*; in such a case, the size of *AM_6* will increase exponentially with respect to the number of added objects. Now we present a rather modular version of the access monitor. The basic idea of this new version (Figure 19) is that every object has a "private" monitor which implements the access functions for such (single) object. To make this, we have decomposed process *Monitor_5* into two different processes, one for each object; then we have composed such processes to their respective objects together with a high level buffer obtaining the *SBSNNI*-secure *Modh* and *Modl* agents. In particular, $Monitor\_7(x)$ handles the accesses to object x ($x = 0$ low, $x = 1$ high). As in *Access_Monitor_6*, we have an interface which guarantees the exclusive use of the monitor within the same level and is able to read values from the high buffer. The resulting system is reported in Table 17 where $L = \{res, access\_r, access\_w\}$ and $Lh = \{r, w, val(1,y)\}$. Table 18 reports the output of the (successful) verification of the *SBSNNI* property for *Access_Monitor_7*. This task takes about 20 seconds on a SUN5 workstation, supporting our claim that a modular definition would help. Moreover, we can also check the new version of the monitor is functionally equivalent to the previous ones: in about 5 minutes, CoSeC is able to check that *Access_Monitor_7* $\approx_B$ *Access_Monitor_5*, and so also *Access_Monitor_7* $\approx_B$ *Access_Monitor_6*.

$$
\begin{aligned}
Access\_Monitor\_7 &\stackrel{\text{def}}{=} (Modh \mid Modl \mid Interf\_6) \setminus L \\
Modh &\stackrel{\text{def}}{=} ((Monitor\_7(1) \mid Object(1,0) \mid hBuf(empty)) \setminus Lh) \\
&\quad [res(0,y)/val(0,y)] \\
Modl &\stackrel{\text{def}}{=} ((Monitor\_7(0) \mid Object(0,0) \mid hBuf(empty)) \setminus Lh) \\
&\quad [res(0,y)/val(0,y)] \\
Monitor\_7(x) &\stackrel{\text{def}}{=} access\_r(l,x). \\
&\quad (\ \textbf{if } x \leq l \textbf{ then} \\
&\qquad r(x,y).\overline{val}(l,y).Monitor\_7(x) \\
&\quad \textbf{else} \\
&\qquad \overline{val}(l,err).Monitor\_7(x)) \\
&\quad + \\
&\quad access\_w(l,x,z). \\
&\quad (\ \textbf{if } x \geq l \textbf{ then} \\
&\qquad \overline{w}(x,z).Monitor\_7(x) \\
&\quad \textbf{else} \\
&\qquad Monitor\_7(x))
\end{aligned}
$$

Table 17. The *Access_Monitor_7*.

In recent papers [28, 3], the underlying model has been extended in order to deal with time and probability. Once an appropriate semantics equivalence has been defined in these new models, the *BNDC* property has been shown to naturally handle the new features of the model. In particular, in such models, *BNDC* has been shown to be able to detect timing and probabilistic convert channels, respectively.

Another aspect we are studying is the possibility of defining a criterion for evaluating the quality of information flow properties [33]. We are trying to do this by defining classes of properties which guarantee the impossibility of the construction of some "canonical" channels. We have seen, for example, that using some systems which are not *BNDC* it is possible to obtain a (initially noisy) perfect channel from high to low level. The aim is to classify the information flow properties depending on which kind of channels they effectively rule out.

We have seen that it is possible to automatically check almost all the properties we have presented. Indeed we are still looking for a good (necessary and sufficient) characterization of the *BNDC* property. We have also briefly presented the CoSeC tool. In [47], Martinelli has applied partial model checking techniques to the verification of *BNDC*, leading to the implementation of an automatic verifier [46] which is able to automatically synthetize the possible interfering high-level process.

As we have stated above, the setting we have proposed is quite general. We claim that information flow (or NI) properties could have a number of different applications since they basically capture the possibility for a class of users of modifying the behaviour of another user class. This generality has allowed to apply some variants of our properties to the analysis of cryptographic protocols [15, 30, 29, 31], starting from a general scheme proposed in [34]. This has been the topic of the second part of the course "Classification of Security Properties" at FOSAD'00 school, and we are presently working on a tutorial which will cover it [27].

This application of NI properties to network security is new to our knowledge. The interesting point is that they can be applied to the verification of protocols with different aims, e.g., authentication, secrecy, key-distribution. We have analyzed a number of different protocols, thanks to a new tool interface which permits to specify value-passing protocols and to automatically generate the enemy [15]; this has also allowed to find new anomalies in some cryptographic protocols [16].

In [19, 20, 11, 12], a new definition of entity authentication, which is based on explicit locations of entities, has been proposed. We are presently trying to characterize also this property through information flow. We also intend to carry the *BNDC* theory over more expressive process calculi, like, e.g., pi/spi-calculus [2] and Mobile Ambients [13]. This would allow to compare it with new recent security properties proposed on such calculi and reminiscent of some Non-Interference ideas (see, e.g., [37, 35]).

20. R. Focardi. "Using Entity Locations for the Analysis of Authentication Protocols". In *Proceedings of Sixth Italian Conference on Theoretical Computer Science (ICTCS'98)*, November 1998.
21. R. Focardi. *Analysis and Automatic Detection of Information Flows in Systems and Networks.* PhD thesis, University of Bologna (Italy), 1999.
22. R. Focardi and R. Gorrieri. "An Information Flow Security Property for CCS". In *Proceedings of the Second North American Process Algebra Workshop (NAPAW '93)*, TR 93-1369, Cornell (Ithaca), August 1993.
23. R. Focardi and R. Gorrieri. "A Taxonomy of Trace-based Security Properties for CCS ". In *Proceedings Seventh IEEE Computer Security Foundation Workshop, (CSFW'94), (Li Gong Ed.)*, pages 126–136, Franconia (NH), June 1994. IEEE Press.
24. R. Focardi and R. Gorrieri. "A Classification of Security Properties for Process Algebras". *Journal of Computer Security*, 3(1):5–33, 1994/1995.
25. R. Focardi and R. Gorrieri. "Automatic Compositional Verification of Some Security Properties". In *Proceedings of Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 167–186, Passau (Germany), March 1996. Springer-Verlag, LNCS 1055.
26. R. Focardi and R. Gorrieri. "The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties". *IEEE Transactions on Software Engineering*, 23(9):550–571, September 1997.
27. R. Focardi, R. Gorrieri, and F. Martinelli. "Classification of Security Properties (Part II: Network Security)". Forthcoming.
28. R. Focardi, R. Gorrieri, and F. Martinelli. "Information Flow Analysis in a Discrete Time Process Algebra". In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW13), (P.Syverson ed.)*, pages 170–184. IEEE CS Press, July 2000.
29. R. Focardi, R. Gorrieri, and F. Martinelli. Message authentication through noninterference. In *Proc. of 8th International Conference in Algebraic Methodology and Software Technology* (AMAST), 2000.
30. R. Focardi, R. Gorrieri, and F. Martinelli. "Non Interference for the Analysis of Cryptographic Protocols". In *Proceedings of ICALP'00*, pages 744–755. LNCS 1853, July 2000.
31. R. Focardi, R. Gorrieri, and F. Martinelli. Secrecy in security protocols as noninterference. In *Workshop on secure architectures and information flow*, volume 32 of *ENTCS*, 2000.
32. R. Focardi, R. Gorrieri, and V. Panini. "The Security Checker: a Semantics-based Tool for the Verification of Security Properties". In *Proceedings Eight IEEE Computer Security Foundation Workshop, (CSFW'95) (Li Gong Ed.)*, pages 60–69, Kenmare (Ireland), June 1995. IEEE Press.
33. R. Focardi, R. Gorrieri, and R. Segala. "A New Definition of Multilevel Security". In *proceedings of Workshop on Issues in the Theory of Security (WITS '00)*, University of Geneva, July 2000.
34. R. Focardi and F. Martinelli. "A Uniform Approach for the Definition of Security Properties". In *Proceedings of World Congress on Formal Methods (FM'99)*, pages 794–813. Springer, LNCS 1708, 1999.
35. C. Fournet and M. Abadi. "Mobile Values, New Names, and Secure Communication". In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, January 2001.

57. P. Y. A. Ryan. "A CSP Formulation of Non-Interference". In *Proceedings of the 1990 Computer Security Foundation Workshop III*, Franconia, 1990. IEEE press.
58. P.Y.A. Ryan. "Mathematical Models of Computer Security". In this volume.
59. S. Schneider. "Verifying authentication protocols in CSP". *IEEE Transactions on Software Engineering*, 24(9), September 1998.
60. G. Smith and D.M. Volpano. "Secure Information Flow in a Multi-Threaded Imperative Language". In *Proc. of POPL*, pages 355–364, 1998.
61. L. J. Stockmeyer and A. R. Meyer. "Word problems requiring exponential time". In *Proceedings of the 5th ACM Symposium on Theory of Computing*, pages 1–9, Austin, Texas, 1973.
62. D. Sutherland. "A Model of Information". In *Proceedings of the 9th National Computer Security Conference*, pages 175–183. National Bureau of Standards and National Computer Security Center, September 1986.
63. C. R. Tsai, V. D. Gligor, and C. S. Chandersekaran. "On the Identification of Covert Storage Channels in Secure Systems". *IEEE Transactions on Software Engineering*, pages 569–580, June 1990.
64. J. T. Wittbold and D. M. Johnson. "Information Flow in Nondeterministic Systems". In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 144–161. IEEE Computer Society Press, 1990.

To appear in *Foundations of Security Analysis and Design*,
R. Focardi and R. Gorrieri (eds), LNCS 2172, Springer-Verlag.

Access Control: Policies, Models, and Mechanisms

Pierangela Samarati[1] and Sabrina De Capitani di Vimercati[2]

[1] Dipartimento di Tecnologie dell'Informazione – Università di Milano
Via Bramante 65 – 26013 - Crema (CR) Italy
samarati@dsi.unimi.it
http//homes.dsi.unimi.it/~samarati
[2] Dip. di Elettronica per l'Automazione – Università di Brescia
Via Branze 38 – 25123 Brescia - Italy
decapita@ing.unibs.it
http//www.ing.unibs.it/~decapita

Abstract. Access control is the process of mediating every request to resources and data maintained by a system and determining whether the request should be granted or denied. The access control decision is enforced by a mechanism implementing regulations established by a security policy. Different access control policies can be applied, corresponding to different criteria for defining what should, and what should not, be allowed, and, in some sense, to different definitions of what ensuring security means. In this chapter we investigate the basic concepts behind access control design and enforcement, and point out different security requirements that may need to be taken into consideration. We discuss several access control policies, and models formalizing them, that have been proposed in the literature or that are currently under investigation.

1 Introduction

An important requirement of any information management system is to *protect data and resources* against unauthorized disclosure (*secrecy*) and unauthorized or improper modifications (*integrity*), while at the same time ensuring their availability to legitimate users (*no denials-of-service*). Enforcing protection therefore requires that *every access to a system and its resources be controlled and that all and only authorized accesses can take place.* This process goes under the name of *access control*. The development of an access control system requires the definition of the regulations according to which access is to be controlled and their implementation as functions executable by a computer system. The development process is usually carried out with a multi-phase approach based on the following concepts:

Security policy: it defines the (high-level) rules according to which access control must be regulated.[1]

[1] Often, the term policy is also used to refer to particular instances of a policy, that is, actual authorizations and access restrictions to be enforced (e.g., Employees can read bulletin-board).

example, from laws, practices, and organizational regulations. A security policy must capture all the different regulations to be enforced and, in addition, must also consider possible additional threats due to the use of a computer system. Access control policies can be grouped into three main classes:

Discretionary (DAC) (authorization-based) policies control access based on the identity of the requestor and on access rules stating what requestors are (or are not) allowed to do.

Mandatory (MAC) policies control access based on mandated regulations determined by a central authority.

Role-based (RBAC) policies control access depending on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles.

Discretionary and role-based policies are usually coupled with (or include) an *administrative* policy that defines who can specify authorizations/rules governing access control.

In this chapter we illustrate different access control policies and models that have been proposed in the literature, also investigating their low level implementation in terms of security mechanisms. In illustrating the literature and the current status of access control systems, of course, the chapter does not pretend to be exhaustive. However, by discussing different approaches with their advantages and limitations, this chapter hopes to give an idea of the different issues to be tackled in the development of an access control system, and of good security principles that should be taken into account in the design.

The chapter is structured as follows. Section 2 introduces the basic concepts of discretionary policies and authorization-based models. Section 3 shows the limitation of authorization-based controls to introduce the basis for the need of mandatory policies, which are then discussed in Section 4. Section 5 illustrates approaches combining mandatory and discretionary principles to the goal of achieving mandatory information flow protection without loosing the flexibility of discretionary authorizations. Section 6 illustrates several discretionary policies and models that have been proposed. Section 7 illustrates role-based access control policies. Finally, Section 8 discusses advanced approaches and directions in the specification and enforcement of access control regulations.

2 Basic concepts of discretionary policies

Discretionary policies enforce access control on the basis of the identity of the requestors and explicit access rules that establish who can, or cannot, execute which actions on which resources. They are called discretionary as users can be given the ability of passing on their privileges to other users, where granting and revocation of privileges is regulated by an administrative policy. Different discretionary access control policies and models have been proposed in the literature. We start in this section with the early discretionary models, to convey the basic ideas of authorization specifications and their enforcement. We will come

	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

Fig. 1. An example of access matrix

describe changes to the state of a system. These operations, whose effect on the authorization state is illustrated in Figure 2, correspond to adding and removing a subject, adding and removing an object, and adding and removing a privilege. Each command has a conditional part and a body and has the form

command $c(x_1, \ldots, x_k)$
if r_1 in $A[x_{s_1}, x_{o_1}]$ and
r_2 in $A[x_{s_2}, x_{o_2}]$ and
.
.
r_m in $A[x_{s_m}, x_{o_m}]$
then op_1
op_2
.
.
op_n
end.

with $n > 0, m \geq 0$. Here $r_1, ..., r_m$ are actions, $op_1, ..., op_n$ are primitive operations, while $s_1, ..., s_m$ and $o_1, ..., o_m$ are integers between 1 and k. If m=0, the command has no conditional part.

For example, the following command creates a file and gives the creating subject ownership privilege on it.

command CREATE(creator,file)
create object file
enter Own *into* A[creator,file] **end.**

The following commands allow an owner to grant to others, and revoke from others, a privilege to execute an action on her files.

command CONFER$_a$(owner,friend,file)
if Own in A[owner,file]
then *enter a into* A[friend,file] **end.**

For instance, a *copy flag*, denoted $*$, attached to a privilege may indicate that the privilege can be transferred to others. Granting of authorizations can then be accomplished by the execution of commands like the one below (again here TRANSFER_a is an abbreviation for as many commands as there are actions).

command TRANSFER_a(subj,friend,file)
  if a^* in A[subj,file]
    then *enter a into* A[friend,file] **end.**

The ability of specifying commands of this type clearly provides flexibility as different administrative policies can be taken into account by defining appropriate commands. For instance, an alternative administrative flag (called *transfer only* and denoted $+$) can be supported, which gives the subject the ability of passing on the privilege to others but for which, so doing, the subject looses the privilege. Such a flexibility introduces an interesting problem referred to as *safety*, and concerned with the propagation of privileges to subjects in the system. Intuitively, given a system with initial configuration Q, the *safety* problem is concerned with determining whether or not a given subject s can ever acquire a given access a on an object o, that is, if there exists a sequence of requests that executed on Q can produce a state Q' where a appears in a cell $A[s,o]$ that did not have it in Q. (Note that, of course, not all leakages of privileges are bad and subjects may intentionally transfer their privileges to "trusworthy" subjects. Trustworthy subjects are therefore ignored in the analysis.) It turns out that the safety problem is undecidable in general (it can be reduced to the halting problem of a Turing machine) [4]. It remains instead decidable for cases where subjects and objects are finite, and in *mono-operational* systems, that is, systems where the body of commands can have at most one operation (while the conditional part can still be arbitrarily complex). However, as noted in [81], mono-operational systems have the limitation of making create operations pretty useless: a single create command cannot do more than adding an empty row/column (it cannot write anything in it). It is therefore not possible to support ownership or control relationships between subjects. Progresses in safety analysis were made in a later extension of the HRU model by Sandhu [81], who proposed the *TAM* (Typed Access Matrix) model. TAM extends HRU with strong typing: each subject and object has a type; the type is associated with the subjects/objects when they are created and thereafter does not change. Safety results decidable in polynomial time for cases where the system is monotonic (privileges cannot be deleted), commands are limited to three parameters, and there are no cyclic creates. Safety remains undecidable otherwise.

2.2 Implementation of the access matrix

Although the matrix represents a good conceptualization of authorizations, it is not appropriate for implementation. In a general system, the access matrix will be usually enormous in size and sparse (most of its cells are likely to be empty). Storing the matrix as a two-dimensional array is therefore a waste of

User	Access mode	Object
Ann	own	File 1
Ann	read	File 1
Ann	write	File 1
Ann	read	File 2
Ann	write	File 2
Ann	execute	Program 1
Bob	read	File 1
Bob	read	File 3
Bob	write	File 3
Carl	read	File 2
Carl	execute	Program 1
Carl	read	Program 1

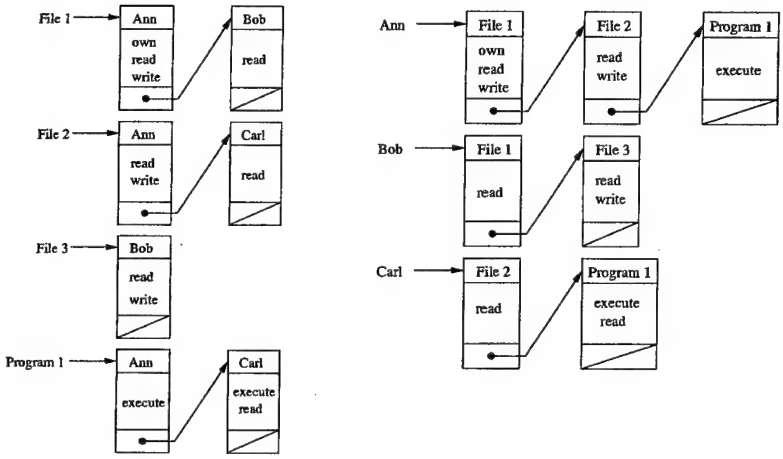


Fig. 3. Authorization table, ACLs, and capabilities for the matrix in Figure 1

Authorizations are represented by associating with each object an access control list of 9 bits: bits 1 through 3 reflect the privileges of the file's owner, bits 4 through 6 those of the user group to which the file belongs, and bits 7 through 9 those of all the other users. The three bits correspond to the read (`r`), write (`w`), and execute (`x`) privilege, respectively. For instance, ACL `rwxr-x--x` associated with a file indicates that the file can be read, written, and executed by its owner, read and executed by users belonging to the group associated with the file, and executed by all the other users.

3 Vulnerabilities of the discretionary policies

In defining the basic concepts of discretionary policies, we have referred to access requests on objects submitted by users, which are then checked againsts the

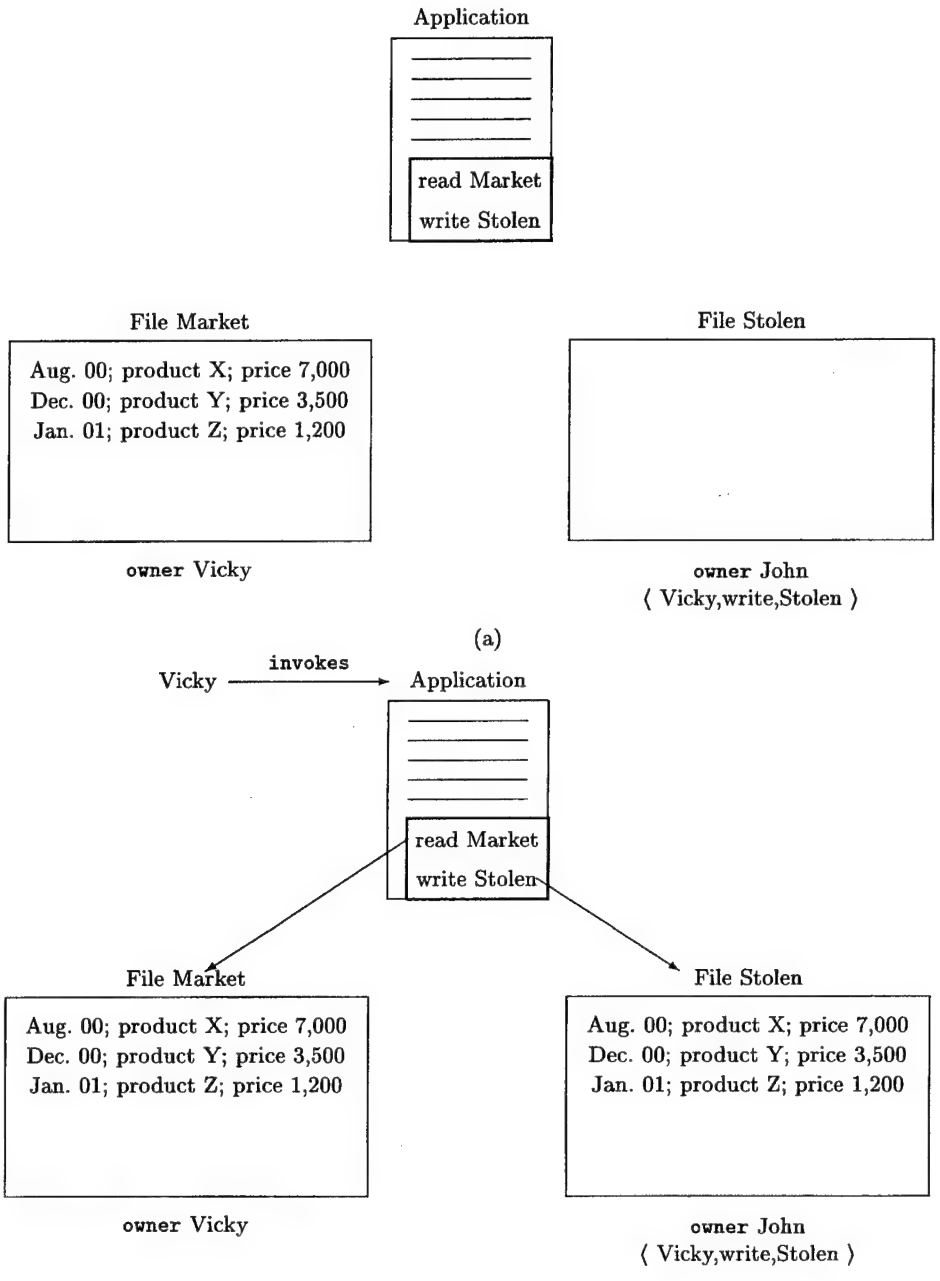


Fig. 4. An example of Trojan Horse improperly leaking information

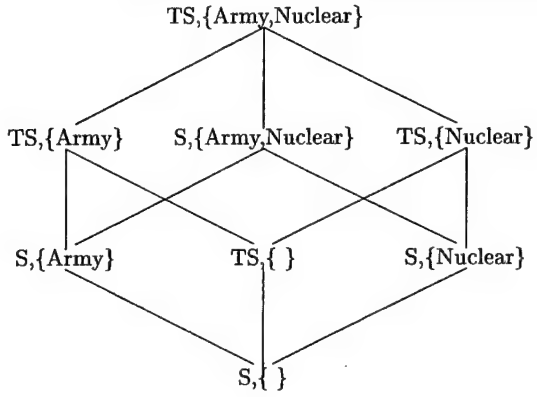


Fig. 5. An example of security lattice

c_2 and the categories of c_1 include those of c_2. Formally, given a totally ordered set of security levels $\mathcal{L}$, and a set of categories $\mathcal{C}$, the set of access classes is $\mathcal{AC} = \mathcal{L} \times \wp(\mathcal{C})^2$, and $\forall c_1 = (L_1, C_1), c_2 = (L_2, C_2) : c_1 \geq c_2 \iff L_1 \geq L_2 \wedge C_1 \supseteq C_2$. Two classes c_1 and c_2 such that neither $c_1 \geq c_2$ nor $c_2 \geq c_1$ holds are said to be *incomparable*.

It is easy to see that the dominance relationship so defined on a set of access classes $\mathcal{AC}$ satisfies the following properties.

- *Reflexivity:* $\forall x \in \mathcal{AC} : x \geq x$
- *Transitivity:* $\forall x, y, z \in \mathcal{AC} : x \geq y, y \geq z \Longrightarrow x \geq z$
- *Antisymmetry:* $\forall x, y \in \mathcal{AC} : x \geq y, y \geq x \Longrightarrow x = y$
- *Existence of a least upper bound:* $\forall x, y \in \mathcal{AC} : \exists\, !z \in \mathcal{AC}$
  - $z \geq x$ and $z \geq y$
  - $\forall t \in \mathcal{AC} : t \geq x$ and $t \geq y \Longrightarrow t \geq z$.
- *Existence of a greatest lower bound:* $\forall x, y \in \mathcal{AC} : \exists\, !z \in \mathcal{AC}$
  - $x \geq z$ and $y \geq z$
  - $\forall t \in \mathcal{AC} : x \geq t$ and $y \geq t \Longrightarrow z \geq t$.

Access classes defined as above together with the dominance relationship between them therefore form a lattice [31]. Figure 5 illustrates the security lattice obtained considering security levels `TS` and `S`, with `TS>S` and the set of categories `{Nuclear,Army}`.

The semantics and use of the classifications assigned to objects and subjects within the application of a multilevel mandatory policy is different depending on whether the classification is intended for a *secrecy* or an *integrity* policy. We next examine secrecy-based and integrity-based mandatory policies.

[2] $\wp(\mathcal{C})$ denotes the powerset of $\mathcal{C}$.

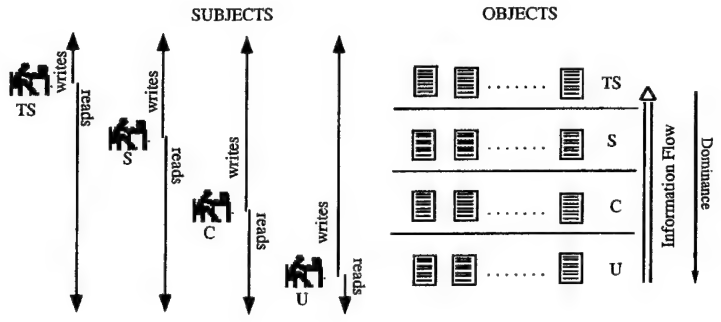


Fig. 6. Information flow for secrecy

Given the no-write-down principle, it is clear now why users are allowed to connect to the system at different access classes, so that they are able to access information at different levels (provided that they are cleared for it). For instance, Vicky has to connect to the system at a level below her clearance if she wants to write some Unclassified information, such as working instructions for John. Note that a lower class does not mean "less" privileges in absolute terms, but only less reading privileges (see Figure 6).

Although users can connect to the system at any level below their clearance, the strict application of the no-read-up and the no-write-down principles may result too rigid. Real world situations often require exceptions to the mandatory restrictions. For instance, data may need to be downgraded (e.g., data subject to embargoes that can be released after some time). Also, information released by a process may be less sensitive than the information the process has read. For instance, a procedure may access personal information regarding the employees of an organization and return the benefits to be granted to each employee. While the personal information can be considered Secret, the benefits can be considered Confidential. To respond to situations like these, multilevel systems should then allow for exceptions, loosening or waiving restrictions, in a controlled way, to processes that are *trusted* and ensure that information is *sanitized* (meaning the sensitivity of the original information is lost).

Note also that DAC and MAC policies are not mutually exclusive, but can be applied jointly. In this case, an access to be granted needs both *i)* the existence of the necessary authorization for it, and *ii)* to satisfy the mandatory policy. Intuitively, the discretionary policy operates *within the boundaries* of the mandatory policy: it can only restrict the set of accesses that would be allowed by MAC alone.

state v_0 is secure, and *ii)* the state transition T is security preserving, that is, it transforms a secure state into another secure state.

As noticed by McLean in his example called "System Z" [63], the BST theorem does not actually guarantee security. The problem lies in the fact that no restriction, but to be preserving of state security, is put on transitions. In his System Z example, McLean shows how failing to control transitions can compromise security. Consider a system Z whose initial state is secure and that has only one type of transition: when a subject requests any type of access to an object o, every subject and object in the system are downgraded to the lowest possible access class and the access is granted. System Z satisfies the Bell and LaPadula notion of security, but it is obviously not secure in any meaningful sense. The problem pointed out by System Z is that transitions need to be controlled. Accordingly, McLean proposes extending the model with a new function $C : S \cup O \rightarrow \wp(S)$, which returns the set of subjects allowed to change the level of its argument. A transition is secure if it allows changes to the level of a subject/object x only by subjects in $C(x)$; intuitively, these are subjects trusted for downgrading. A system (v_0, R, T) is *secure* if and only if *i)* v_0 is secure, *ii)* every state reachable from v_0 by executing a finite sequence of one or more requests from R is (BLP) secure, and *iii)* T *is transition secure.*

The problem with changing the security level of subjects and objects was not captured formally as an axiom or property in the Bell and LaPadula, but as an informal design guidance called *tranquility* principle. The tranquility principle states that the classification of active objects should not be changed during normal operation [55]. A subsequent revision of the model [10] introduced a distinction between the level assigned to a subject (*clearance*) and its current level (which could be any level dominated by the clearance), which also implied changing the formulation of the axioms, introducing more flexibility in the control.

Another property included in the Bell and LaPadula model is the *discretionary property* which constraints the set of current accesses b to be a subset of the access matrix M. Intuitively, it enforces discretionary controls.

4.4 Integrity-based mandatory policies: The Biba model

The mandatory policy that we have discussed above protects only the confidentiality of the information; no control is enforced on its integrity. Low classified subjects could still be able to enforce improper indirect modifications to objects they cannot write. With reference to our organization example, for instance, integrity could be compromised if the Trojan Horse implanted by John in the application would write data in file Market (this operation would not be blocked by the secrecy policy). Starting from the principles of the Bell and LaPadula model, Biba [16] proposed a dual policy for safeguarding integrity, which controls the flow of information and prevents subjects to *indirectly* modify information they cannot write. Like for secrecy, each subject and object in the system is assigned an integrity classification. The classifications and the dominance relationship between them are defined as before. Example of integrity levels can be: Crucial

ever, if a subject s writes an object o, the object has its classification downgraded to the greatest lower bound of the classification of the two, that is, $\lambda'(o) = \mathsf{glb}(\lambda(s), \lambda(o))$.

Intuitively, the two policies attempt to apply a more dynamic behavior in the enforcement of the constraints. The two approaches suffer however of drawbacks. In the low-water mark for subjects approach, the ability of a subject to execute a procedure may depend on the order with which operations are requested: a subject may be denied the execution of a procedure because of read operations executed before. The latter policy cannot actually be considered as safeguarding integrity: given that subjects are allowed to write above their level, integrity compromises can certainly occur; by downgrading the level of the object the policy simply signals this fact.

As it is visible from Figures 6 and 7, secrecy policies allow the flow of information only from lower to higher (secrecy) classes while integrity policies allow the flow of information only from higher to lower (integrity) classes. If both secrecy and integrity have to be controlled, objects and subjects have to be assigned two access classes, one for secrecy control and one for integrity control.

A major limitation of the policies proposed by Biba is that they only capture integrity compromises due to impoproper information flows. However, integrity is a much broader concept and additional aspects should be taken into account (see Section 6.5).

4.5 Applying mandatory policies to databases

The first formulation of the multilevel mandatory policies, and the Bell LaPadula model, simply assumed the existence of objects (information container) to which a classification is assigned. This assumption works well in the operating system context, where objects to be protected are essentially files containing the data. Later studies investigated the extension of mandatory policies to database systems. While in operating systems access classes are assigned to files, database systems can afford a finer-grained classification. Classification can in fact be considered at the level of relations (equivalent to file-level classification in OS), at the level of columns (different properties can have a different classification), at the level of rows (properties referred to a given real world entity or association have the same classification), or at the level of single cells (each data element, meaning the value assigned to a property for a given entity or association, can have a different classification), this latter being the finest possible classification. Early efforts to classifying information in database systems, considered classification at the level of each single element [50, 61]. Element-level classification is clearly appealing since it allows the assignment of a security class to each single real world fact that needs to be represented. For instance, an employee's name can be labeled Unclassified, while his salary can be labeled Secret; also the salary of different employees can take on different classifications. However, the support of fine-grained classifications together with the obvious constraint of

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Jim	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S
Ann	U	Dept1	U	100K	U
Sam	U	Dept1	U	100K	U

(a)

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Jim	U	Dept1	U	100K	U
Ann	U	Dept1	U	100K	U
Sam	U	Dept1	U	100K	U

(b)

Fig. 9. An example of a relation with polyinstantiation (a) and the Unclassified view on it (b)

convey information, the Unclassified subject should see no difference between values that are actually `null` in the database and those that are `null` since they have a higher classification.[5] To produce a view consistent with the relational database constraints the classification needs to satisfy at least the following two basic constraints: *i)* the key attributes must be uniformly classified, and *ii)* the classifications of nonkey attributes must dominate that of key attributes. If it were not so, the view at some levels would contain a null value for some or all key attributes (and therefore would not satisfy the key constraints).

To see how polyinstantiation can arise, suppose that an Unclassified subject, whose view on the table in Figure 8(a) is as illustrated in Figure 8(b), requests insertion of tuple `(Ann, Dept1, 100K)`. According to the key constraints imposed by the relational model, no two tuples can have the same value for the key attributes. Therefore if classifications were not taken into account, the insertion could have not been accepted. The database could have two alternative choices: *i)* tell the subject that a tuple with the same key already exists, or *ii)* replace the old tuple with the new one. The first solution introduces a *covert channel*[6], since by rejecting the request the system would be revealing protected information (meaning the existence of a Secret entity named `Ann`), and clearly compromises secrecy. On the other hand, the second solution compromises integrity, since high classified data would be lost, being overridden by the newly inserted tuple. Both solutions are therefore inapplicable. The only remaining solution would then be to accept the insertion and manage the presence of both tuples (see Figure 9(a)). Two tuples would then exist with the same value, but different classification, for their key (*polyinstantiated tuples*). A similar situation happens if the unclassified subject requests to update the salary of `Sam` to value `100K`. Again, telling the subject that a value already exists would compromise secrecy (if the subject is not suppose to distinguish between real nulls and values

[5] Some proposals do not adopt this assumption. For instance, in LDV [43], a special value "restricted" appears in a subject's view to denote the existence of values not visible to the subject.

[6] We will talk more about covert channels in Section 4.6.

only at a higher level). The presence of two tuples with the same key and same key classification but that differ for the value and classification of some of its attributes can be interpreted as a *single* real world entity for which different values are recorded (corresponding to the different beliefs at different levels). However, unfortunately, polyinstantiation quickly goes out of hand, and the execution of few operations could result in a database whose semantics does not appear clear anymore. Subsequent work tried to establish constraints to maintain semantic integrity of the database status [69, 75, 90]. However, probably because of all the complications and semantics confusion that polyinstantiation bears, fine-grained multilevel databases did not have much success, and current DBMSs do not support element-level classification. Commercial systems (e.g., Trusted Oracle [66] and SYBASE Secure SQL Server) support tuple level classification.

It is worth noticing that, although polyinstantiation is often blamed to be the reason why multilevel relational databases did not have success, polyinstantiation is not necessarily always bad. Controlled polyinstantiation may, for example, be useful to support *cover stories* [38, 49], meaning non-true data whose presence in the database is meant to hide the existence of the actual value. Cover stories are useful when the fact that a given data is not released is by itself a cause of information leakage. For instance, suppose that a subject requires access to a hospital's data and the hospital returns, for all its patients, but for few of them, the illness for which they are being cured. Suppose also that HIV never appears as an illness value. Observing this, the recipient may infer that it is probably the case that the patients for which illness is not disclosed suffer from HIV. The hospital could have avoided exposure to such an inference by simply releasing a non-true alternative value (*cover story*) for these patients. Intuitively, cover stories are "lies" that the DBMS says to uncleared subjects not to disclose (directly or indirectly) the actual values to be protected. We do note that, while cover stories are useful for protection, they have raise objections for the possible integrity compromises which they may indirectly cause, as low level subjects can base their actions on cover stories they believe true.

A complicating aspects in the support of a mandatory policy at a fine-grained level is that the definition of the access class to be associated with each piece of data is not always easy [30]. This is the case, for example, of *association* and *aggregation* requirements, where the classification of a set of values (properties, resp.) is higher than the classification of each of the values singularly taken. As an example, while names and salaries in an organization may be considered Unclassified, the association of a specific salary with an employee's name can be considered Secret (association constraint). Similarly, while the location of a single military ship can be Unclassified, the location of all the ships of a fleet can be Secret (aggregation constraint), as by knowing it one could infer that some operations are being planned. Proper data classification assignment is also complicated by the need to take into account possible inference channels [30, 47, 59]. There is an inference channel between a set of data x and a set of data y if, by knowing x a user can infer some information on y (e.g., an inference channel can exist between an employee's taxes and her salary). Inference-aware

only *overt* channels of information (i.e., flow through *legitimate* channels); they still remain vulnerable to *covert channels*. Covert channels are channels that are not intended for normal communication, but still can be exploited to infer information. For instance, consider the request of a low level subject to write a non-existent high level file (the operation is legitimate since write-up operations are allowed). Now, if the system returns the error, it exposes itself to improper leakages due to malicious high level processes creating and destroying the high level file to signal information to low processes. However, if the low process is not informed of the error, or the system automatically creates the file, subjects may not be signalled possible errors made in legitimate attempts to write. As another example, consider a low level subject that requires a resource (e.g., CPU or lock) that is busy by a high level subject. The system, by not allocating the resource because it is busy, can again be exploited to signal information at lower levels (high level processes can module the signal by requiring or releasing resources). If a low process can see any different result due to a high process operation, there is a channel between them. Channels may also be enacted without modifying the system's response to processes. This is, for example, the case of *timing channels*, that can be enacted when it is possible for a high process to affect the system's response time to a low process. With timing channels the response that the low process receives is always the same, it is the time at which the low process receives the response that communicates information. Therefore, in principle, any *common resource or observable property* of the system state can be used to leak information. Consideration of covert channels requires particular care in the design of the enforcement mechanism. For instance, locking and concurrency mechanisms must be revised and be properly designed [7]. A complication in their design is that care must be taken to avoid the policy for blocking covert channels to introduce denials-of-service. For instance, a trivial solution to avoid covert channels between high and low level processes competing over common resources could be to always give priority to low level processes (possibly terminating high level processes). This approach, however, exposes the systems to denials-of-service attacks whereby low level processes can impede high level (and therefore, presumably, more important) processes to complete their activity.

Covert channels are difficult to control also because of the difficulty of mapping an access control model's primitive to a computer system [64]. For this reason, covert channels analysis is usually carried out in the implementation phase, to make sure that the implementation of the model's primitive is not too weak. Covert channel analysis can be based on tracing the information flows in programs [31], checking programs for shared resources that can be used to transfer information [52], or checking the system clock for timing channels [92]. Beside the complexity, the limitation of such solutions is that covert channels are found out at the end of the development process, where system changes are much more expensive to correct. Interface models have been proposed which attempt to rule out covert channels analysis in the modeling phase [64, 37]. Rather than specifying a particular method to enforce security, interface models specify restrictions on a system's input/output that must be obeyed to avoid covert

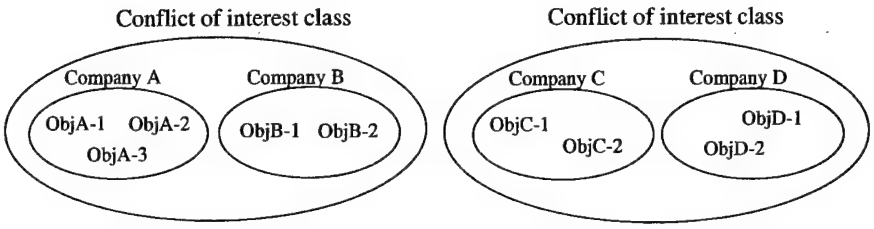


Fig. 12. An example of object organization

– belongs to an entirely different conflict of interest class.

***-property** Write access is only permitted if

– access is permitted by the simple security rule, and
– no object can be read which *i)* is in a different company dataset than the one for which write access is requested, and *ii)* contains unsanitized information.

The term subject used in the properties is to be interpreted as user (meaning access restrictions are referred to users). The reason for this is that, unlike mandatory policies that control processes, the Chinese Wall policy controls users. It would therefore not make sense to enforce restrictions on processes as a user could be able to acquire information about organizations that are in conflict of interest simply running two different processes.

Intuitively, the simple security rule blocks direct information leakages that can be attempted by a single user, while the *-property blocks indirect information leakages that can occur with the collusion of two or more users. For instance, with reference to Figure 12, an indirect improper flow could happen if, *i)* a user reads information from object ObjA-1 and writes it into ObjC-1, and subsequently *ii)* a different user reads information from ObjC-1 and writes it into ObjB-1.

Clearly, the application of the Chinese Wall policy still has some limitations. In particular, strict enforcement of the properties may result too rigid and, like for the mandatory policy, there will be the need for exceptions and support of sanitization (which is mentioned, but not investigated, in [22]). Also, the enforcement of the policies requires keeping and querying the history of the accesses. A further point to take into consideration is to ensure that the enforcement of the properties will not block the system working. For instance, if in a system composed of ten users there are eleven company datasets in a conflict of interest class, then one dataset will remain inaccessible. This aspect was noticed in [22], where the authors point out that there must be at least as many users as the maximum number of datasets which appear together in a conflict of interest class. However, while this condition makes the system operation possible, it cannot ensure it when users are left completely free choice on the datasets

access attributes therefore control information flow. When a new value of some object y is produced as a function of objects in $x_1, \ldots, x_n$, then the potential access attribute of y is set to be the intersection of the potential access attributes of $x_1, \ldots, x_n$.

Walter et al. [87] propose an interpretation of the mandatory controls within the discretionary context. Intuitively, the policy behind this approach, which we call *strict* policy, is based on the same principles as the mandatory policy. Access control lists are used in place of labels, and the inclusion relationship between sets is used in place of the dominance relationship between labels. Information flow restrictions impose that a process can write an object o only if o is protected in reading at least as all the objects read by the process up to that point. (An object o is at least as protected in reading as another object o' if the set of subjects allowed to read o is contained in the set of subjects allowed to read o'.) Although the discretionary flexibility of specifying accesses is not lost, the overall flexibility is definitely reduced by the application of the strict policy. After having read an object o, a process is completely unable to write any object less protected in reading than o, even if the write operation would not result in any improper information leakage.

Bertino et al. [14] present an enhancement of the strict policy to introduce more flexibility in the policy enforcement. The proposal bases on the observation that whether or not some information can be released also depends on the procedure enacting the release. A process may access sensitive data and yet not release any sensitive information. Such a process should be allowed to bypass the restrictions of the strict policy, thus representing an *exception*. On the other side, the information produced by a process may be more sensitive than the information the process has read. An exception should in this case restrict the write actions otherwise allowed by the strict policy. Starting from these observations, Bertino et al. [14] allow procedures to be granted exceptions to the strict policy. The proposal is developed in the context of object-oriented systems, where the modularity provided by methods associated with objects allows users to identify specific pieces of *trusted* code for which exceptions can be allowed, and therefore provide flexibility in the application of the control. Exceptions can be positive or negative. A positive exception overrides a restriction imposed by the strict policy, permitting an information flow which would otherwise be blocked. A negative exception overrides a permission stated by the strict policy forbidding an information flow which would otherwise be allowed. Two kinds of exceptions are supported by the model: *reply-exceptions* and *invoke-exceptions*. Reply exceptions apply to the information returned by a method. Intuitively, positive reply exceptions apply when the information returned by a method is less sensitive than the information the method has read. Reply exceptions can waive the strict policy restrictions and allow information returned by a method to be disclosed to users not authorized to read the objects that the method has read. Invoke exceptions apply during a method's execution, for write operations that the method requests. Intuitively, positive invoke exceptions apply to methods that are trusted not to leak (through write operations or method invocations) the

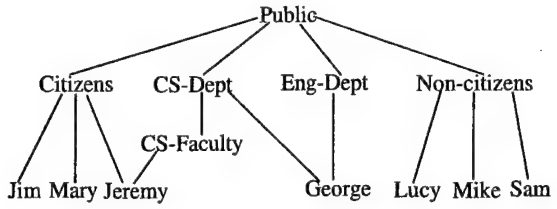


Fig. 13. An example of user-group hierarchy

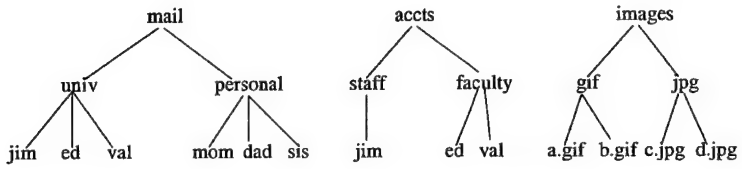


Fig. 14. An example of object hierarchy

reflect the logical file system tree structure, while in object-oriented system it can reflect the class (is-a) hierarchy. Figure 14 illustrates an example of object hierarchy. Even actions can be organized hierarchically, where the hierarchy may reflect an implication of privileges (e.g., write is more powerful than read [70]) or a grouping of sets of privileges (e.g., a "writing privileges" group can be defined containing write, append, and undo [84]). These hierarchical relationships can be exploited *i)* to support preconditions on accesses (e.g., in Unix a subject needs the execute, x, privilege on a directory in order to access the files within it), or *ii)* to support authorization implication, that is, authorizations specified on an abstraction apply to all its members. Support of abstractions with implications provides a short hand way to specify authorizations, clearly simplifying authorization management. As a matter of fact, in most situations the ability to execute privileges depends on the membership of users into groups or objects into collections: translating these requirements into basic triples of the form (user,object,action) that then have to be singularly managed is a considerable administrative burden, and makes it difficult to maintain both satisfactory security and administrative efficiency. However, although there are cases where abstractions can work just fine, many will be the cases where exceptions (i.e., authorizations applicable to all members of a group but few) will need to be supported. This observation has brought to the combined support of both *positive* and *negative* authorizations. Traditionally, positive and negative authorizations have been used in mutual exclusion corresponding to two classical approaches to access control, namely:

- what if for two authorizations the most specific relationship appear reversed over different domains? For instance, consider authorizations (CS-Faculty, read+, mail) and (CS-Dept, read−, personal); the first has a more specific subject, while the second has a more specific object (see Figures 13 and 14).

A slightly alternative policy on the same line as the most specific policy is what in [48] is called *most-specific-along-a-path-takes-precedence*. This policy considers an authorization specified on an element x as overriding an authorization specified on a more general element y only for those elements that are members of y because of x. Intuitively, this policy takes into account the fact that, even in the presence of a more specific authorization, the more general authorization can still be applicable because of other paths in the hierarchy. For instance, consider the group hierarchy in Figure 13 and suppose that for an access a positive authorization is granted to Public while a negative authorization is granted to CS-Dept. What should we decide for George? On the one side, it is true that CS-Dept is more specific than Public; on the other side, however, George belongs to Eng-Dept, and for Eng-Dept members the positive authorization is not overridden. While the most-specific-takes-precedence policy would consider the authorization granted to Public as being overridden for George, the most-specific-along-a-path considers both authorizations as applicable to George. Intuitively, in the most-specific-along-a-path policy, an authorization propagates down the hierarchy until overridden by a more specific authorization [35].

The most specific argument does not always apply. For instance, an organization may want to be able to state that consultants should not be given access to private projects, *no exceptions allowed*. However, if the most specific policy is applied, any authorization explicitly granted to a single consultant will override the denial specified by the organization. To address situations like this, some approaches proposed adopting *explicit priorities*. In ORION [70], authorizations are classified as *strong* or *weak*: weak authorizations override each other based on the most-specific policy, and strong authorizations override weak authorizations (no matter their specificity) and *cannot be overridden*. Given that strong authorizations must be certainly obeyed, they are required to be consistent. However, this requirement may be not always be enforceable. This is, for example, the case where groupings are not explicitly defined but depend on the evaluation of some conditions (e.g., "all objects owned by Tom", "all objects created before 1/1/01"). Also, while the distinction between strong and weak authorizations is convenient in many situations and, for example, allows us to express the organizational requirement just mentioned, it is limited to two levels of priority, which may not be enough. Many other conflict resolution policies can be applied. Some approaches, extending the strong and weak paradigm, proposed adopting *explicit priorities*; however, these solutions do not appear viable as the authorization specifications may result not always clear. Other approaches (e.g., [84]) proposed making authorization priority dependent on the *order in which authorizations are listed* (i.e., the authorizations that is encountered first applies). This approach, however, has the drawback that granting or removing an au-

specifications. However, the complications brought by negative authorizations are not due to negative authorizations themselves, but to the different semantics that the presence of permissions and denials can have, that is, to the complexity of the different real world scenarios and requirements that may need to be captured. There is therefore a trade-off between expressiveness and simplicity. For this reason, most current systems adopting negative authorizations for exception support impose specific conflict resolution policies, or support a limited form of conflict resolution. For instance, in the Apache server [6], authorizations can be positive and negative and an ordering ("deny,allow" or "allow,deny") can be specified dictating how negative and positive authorizations are to be interpreted. In the "deny,allow" order, negative authorizations are evaluated first and access is allowed by default (open policy). Any client that does not match a negative authorization or matches a positive authorization is allowed access. In the "allow,deny" order, the positive authorizations are evaluated first and access is denied by default (closed policy). Any client that does not match a positive authorization *or* does match a negative authorization will be denied access.

More recent approaches are moving towards the development of flexible frameworks with the support of multiple conflict resolution and decision policies. We will examine them in Section 8.

Other advancements in authorization specification and enforcement have been carried out with reference to specific applications and data models. For instance, authorization models proposed for object-oriented systems (e.g., [2, 35, 71]) exploit the *encapsulation* concept, meaning the fact that access to objects is always carried out through methods (read and write operations being primitive methods). In particular, users granted authorizations to invoke methods can be given the ability to successfully complete them, without need to have the authorizations for all the accesses that the method execution entails. For instance, in OSQL, each derived function (i.e., method) can be specified as supporting *static* or *dynamic* authorizations [2]. A dynamic authorization allows the user to invoke the function, but its successful completion requires the user to have the authorization for all the calls the function makes during its execution. With a *static* authorization, calls made by the function are checked against the creator of the function, instead of those of the calling user. Intuitively, static authorizations behave like the setuid (set user id) option, provided by the Unix operating system that, attached to a program (e.g., lpr) implies that all access control checks are to be performed against the authorizations of the program's owner (instead of those of the caller as it would otherwise be). A similar feature is also proposed in [71], where each method is associated with a principal, and accesses requested during a method execution are checked against the authorization of the method's principal. Encapsulation is also exploited by the Java 2 security model [83] where authorizations can be granted to code, and requests to access resources are checked against the authorizations of the code directly attempting the access.

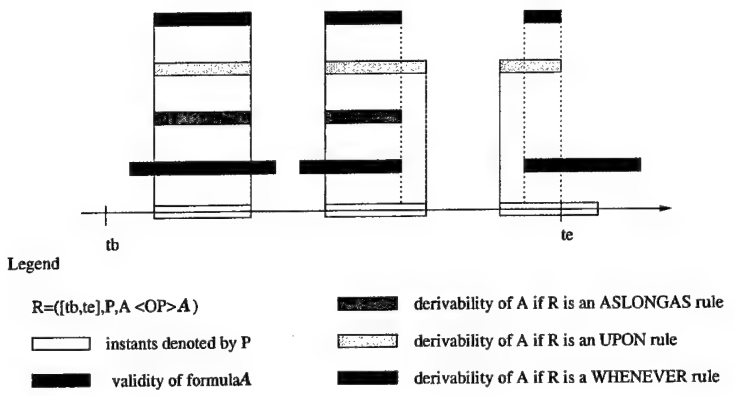


Fig. 16. Semantics of the different temporal operators [13]

guarantees efficient access. The model is focussed on time-based constraints and reasoning and allows expressing authorization relationships and derivation not covered in other models. However, it does not address the enforcement of different implication and conflict resolution policies (conflicts between permissions and denials are solved according to the denials-take-precedence policy).

6.3 A calculus for access control

Abadi et al. [1] present a calculus for access control that combines authentication (i.e., identity check) and authorization control, taking also into account possible delegation of privileges among parties. The calculus is based on the notion of *principals*. Principals are sources of requests and make statements (e.g., "read file tmp"). Principals can be either simple (e.g., users, machines, and communication channels) or composite. Composite principals are obtained combining principals by means of constructors that allow to capture groups and delegations.a Principals can be as follows [1]:

- *Users* and *machines*.
- *Channels*, such as input devices and cryptographic channels.
- *Conjunction of principals*, of the form $A \wedge B$. A request from $A \wedge B$ is a request that both A and B make (it is not necessary that the request be made in concert).
- *Groups*, define groups of principals, membership of principal P_i in group G_i is written $P_i \Longrightarrow G_i$. Disjunction $A \vee B$ denotes a group composed only of A and B.

- *Ownership:* Each object is associated with an owner, who generally coincides with the user who created the object. Users can grant and revoke authorizations on the objects they own.
- *Decentralized:* Extending the previous approaches, the owner of an object (or its administrators) can delegate other users the privilege of specifying authorizations, possibly with the ability of further delegating it.

Decentralized administration is convenient since it allows users to delegate administrative privileges to others. Delegation, however, complicates the authorization management. In particular, it becomes more difficult for users to keep track of who can access their objects. Furthermore, revocation of authorizations becomes more complex. There are many possible variations on the way decentralized administration works, which may differ in the way the following questions are answered.

- what is the granularity of administrative authorizations?
- can delegation be restricted, that is, can the grantor of an administrative authorization impose restrictions on the subjects to which the recipient can further grant the authorization?
- who can revoke authorizations?
- what about authorizations granted by the revokee?

In general, existing decentralized policies allow users to grant administration for a specific privilege (meaning a given access on a given object). They do not allow, however, to put constraints on the subjects to which the recipient receiving administrative authority can grant the access. This feature could, however, result useful. For instance, an organization could delegate one of its employees to granting access to some resources constraining the authorizations she can grant to employees working within her laboratory. Usually, authorizations can be revoked only by the user who granted them (or, possibly, by the object's owner). When an administrative authorization is revoked, the problem arises of dealing with the authorizations specified by the users from whom the administrative privilege is being revoked. For instance, suppose that Ann gives Bob the authorization to read File1 and gives him the privilege of granting this authorization to others (in some systems, such capability of delegation is called *grant option* [42]). Suppose then that Bob grants the authorization to Chris, and susequently Ann revokes the authorization from Bob. The question now is: what should happen to the authorization that Chris has received? To illustrate how revocation can work it is useful to look at the history of System R [42]. In the System R authorization model, users creating a table can grant other users access privileges on it. Authorizations can be granted with the *grant-option*. If a user receives the authorization for an access with the grant-option she can grant the access (and the grant option on it) to others. Intuitively, this introduces a chain of authorizations. The original System R policy, which we call *(time-based) cascade* revocation, adopted the following semantics for revocation: when a user is revoked the grant option on an access, all authorizations that

the organization. Suppose there is a change in the task or function of a user (say, because of a job promotion). This change may imply a change in the responsibilities of the user and therefore in her privileges. New authorizations will be granted to the user and some of her previous authorizations will be revoked. Applying a recursive revocation will result in the undesirable effect of deleting all authorizations the revokee granted and, recursively, all the authorizations granted through them, which then will need to be re-issued. Moreover, all application programs depending on the revoked authorizations will be invalidated. An alternative form of revocation was proposed in [15], where *non-cascade* revocation is introduced. Instead of deleting all the authorizations granted by the revokee in virtue of the authorizations being revoked, non-recursive revocation re-specifies them to be under the authority of the revoker, which can then retain or selectively delete them. The original time-based revocation policy of System R, was changed to not consider time anymore. In SQL:1999 [28] revocation can be requested *with* or *without cascade*. Cascade revocation recursively deletes authorizations if the revokee does not hold anymore the grant option for the access. However, if the revokee still holds the grant option for the access, the authorizations she granted are not deleted (regardless of time they were granted). For instance, with reference to Figure 17(a), the revocation by Bob of the authorization granted to David, would only delete the authorization granted to David by Bob. Ellen's authorization would still remain valid since David still holds the grant option of the access (because of the authorization from Chris). With the non cascade option the system rejects the revoke operation if its enforcement would entail deletion of other authorizations beside the one for which revocation is requested.

6.5 Integrity policies

In Section 4.4 we illustrated a mandatory policy (namely Biba's model) for protecting information integrity. Biba's approach, however, suffers of two major drawbacks: *i)* the constraints imposed on the information flow may result too restrictive, and *ii)* it only controls integrity intended as the prevention of a flow of information from low integrity objects to high integrity objects. However, this notion of one-directional information flow in a lattice captures only a small part of the data integrity problem [74].

Integrity is concerned with ensuring that no resource (including data and programs[9]) has been modified in an *unauthorized* or *improper* way and that the data stored in the system correctly reflect the real world they are intended to represent (i.e., that users expect). Integrity preservation requires prevention of frauds and errors, as the term "improper" used above suggests: violations to data integrity are often enacted by legitimate users executing authorized actions but misusing their privileges.

Any data management system today has functionalities for ensuring integrity [8]. Basic integrity services are, for example, *concurrency control* (to

[9] Programs improperly modified can fool the access control and bypass the system restrictions, thus violating the secrecy and/or integrity of the data (see Section 3).

C1: All IVPs must ensure that all CDIs are in a valid state when the IVP is run.
C2: All TPs must be certified to be valid (i.e., preserve validity of CDIs' state)
C3: Assignment of TPs to users must satisfy separation of duty
C4: The operations of TPs must be logged
C5: TPs execute on UDIs must result in valid CDIs
E1: Only certified TPs can manipulate CDIs
E2: Users must only access CDIs by means of TPs for which they are authorized
E3: The identity of each user attempting to execute a TP must be authenticated
E4: Only the agent permitted to certify entities can change the list of such entities associated with other entities

Fig. 18. Clark and Wilson integrity rules

- *Constrained Data Items.* CDIs are the objects whose integrity must be safeguarded.
- *Unconstrained Data Items.* UDIs are objects that are not covered by the integrity policy (e.g., information typed by the user on the keyboard).
- *Integrity Verification Procedures.* IVPs are procedures meant to verify that CDIs are in a valid state, that is, the IVPs confirm that the data conforms to the integrity specifications at the time the verification is performed.
- *Transformation Procedures.* TPs are the only procedures (well-formed procedures) that are allowed to modify CDIs or to take arbitrary user input and create new CDIs. TPs are designed to take the system from one valid state to the next

Intuitively, IVPs and TPs are the means for enforcing the well-formed transaction requirement: all data modifications must be carried out through TPs, and the result must satisfy the conditions imposed by the IVPs.

Separation of duty must be taken care of in the definition of authorized operations. In the context of the Clark and Wilson's model, authorized operations are specified by assigning to each user a set of well-formed transactions that she can execute (which have access to constraint data items). Separation of duty requires the assignment to be defined in a way that makes it impossible for a user to violate the integrity of the system. Intuitively, separation of duty is enforced by splitting operations in subparts, each to be executed by a different person (to make frauds difficult). For instance, any person permitted to create or certify a well-formed transaction should not be able to execute it (against production data).

Figure 18 summarizes the nine rules that Clark and Wilson presented for the enforcement of system integrity. The rules are partitioned into two types: certification (C) and enforcement (E). Certification rules involve the evaluation of transactions by an administrator, whereas enforcement is performed by the system.

The Clark and Wilson's proposal outlines good principles for controlling integrity. However, it has limitations due to the fact that it is far from formal and it is unclear how to formalize it in a general setting.

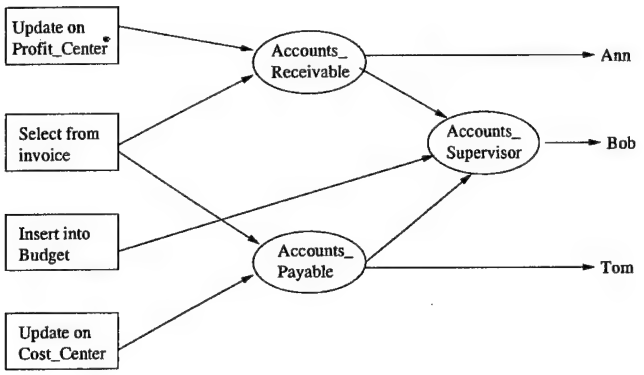


Fig. 19. An example of NPD privilege graph [9]

Although different proposals have been made (e.g., [3, 36, 45, 56, 67, 76, 80]), the basic concepts are common to all approaches. Essentially, role based policies require the identification of *roles* in the system, where a role can be defined as a set of actions and responsibilities associated with a particular working activity. The role can be widely scoped, reflecting a user's job title (e.g., `secretary`), or it can be more specific, reflecting, for example, a task that the user needs to perform (e.g., `order_processing`). Then, instead of specifying all the accesses each users is allowed to execute, access authorizations on objects are specified for roles. Users are then given authorizations to adopt roles (see Figure 20). The user playing a role is allowed to execute all accesses for which the role is authorized. In general, a user can take on different roles on different occasions. Also the same role can be played by several users, perhaps simultaneously. Some proposals for role-based access control (e.g., [76, 80]) allow a user to exercise multiple roles at the same time. Other proposals (e.g., [28, 48]) limit the user to only one role at a time, or recognize that some roles can be jointly exercised while others must be adopted in exclusion to one another. It is important to note the difference between groups (see Section 6) and roles: groups define sets of users while roles define sets of privileges. There is a semantic difference between them: roles can be "activated" and "deactivated" by users at their discretion, while group membership always applies, that is, users cannot enable and disable group memberships (and corresponding authorizations) at their will. However, since roles can be defined which correspond to organizational figures (e.g., `secretary`, `chair`, and `faculty`), a same "concept" can be seen both as a group and as a role.

The role-based approach has several advantages. Some of these are discussed below.

Authorization management Role-based policies benefit from a logical independence in specifying user authorizations by breaking this task into two parts: *i)* assignement of roles to users, and *ii)* assignement of authorizations

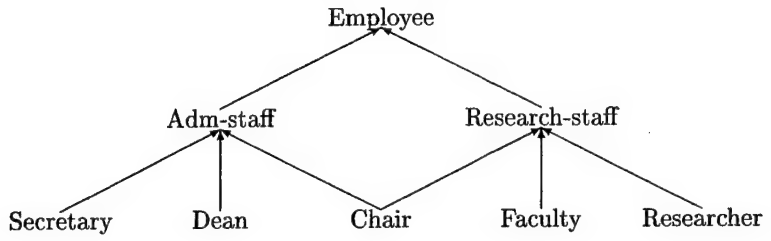


Fig. 21. An example of role hierarchy

Separation of duties Separation of duties refer to the principle that no user should be given enough privileges to misuse the system on their own. For instance, the person authorizing a paycheck should not be the same person who can prepare them. Separation of duties can be enforced either statically (by defining conflicting roles, that is, roles which cannot be executed by the same user) or dynamically (by enforcing the control at access time). An example of dynamic separation of duty restriction is the two-person rule. The first user to execute a two-person operation can be any authorized user, whereas the second user can be any authorized user different from the first [79].

Constraints enforcement Roles provide a basis for the specification and enforcement of further protection requirements that real world policies may need to express. For instance, cardinality constraints can be specified, that restrict the number of users allowed to activate a role or the number of roles allowed to exercise a given privilege. The constraints can also be dynamic, that is, be imposed on roles activation rather than on their assignment. For instance, while several users may be allowed to activate role `chair`, a further constraint can require that at most one user at a time can activate it.

Role-based policies represent a promising direction and a useful paradigm for many commercial and government organizations. However, there is still some work to be done to cover all the different requirements that real world scenarios may present. For instance, the simple hierarchical relationship as intended in current proposals may not be sufficient to model the different kinds of relationships that can occur. For example, a secretary may need to be allowed to write specific documents *on behalf* of her manager, but neither role is a specialization of the other. Different ways of propagating privileges (delegation) should then be supported. Similarly, administrative policies should be enriched. For instance, the traditional concept of ownership may not apply anymore: a user does not necessarily own the objects she created when in a given role. Also, users' identities should not be forgotten. If it true that in most organizations, the role (and not the identity) identifies the privileges that one may execute, it is also true that in some cases the requestor's identity needs to be considered even when a role-based policy is adopted. For instance, a doctor may be allowed to specify treatments and access files but she may be restricted to treatments and files

flexibility and extensibility in access specifications and illustrates how these advantages can be achieved by abstracting from the low level authorization triples and adopting a high level authorization language. Their language is essentially a many-sorted first-order language with a rule construct, useful to express authorization derivations and therefore model authorization implications and default decisions (e.g., closed or open policy). The use of a very general language, which has almost the same expressive power of first order logic, allows the expression of different kinds of authorization implications, constraints on authorizations, and access control policies. However, as a drawback, authorization specifications may result difficult to understand and manage. Also, the trade-off between expressiveness and efficiency seems to be strongly unbalanced: the lack of restrictions on the language results in the specification of models which may not even be decidable and therefore will not be implementable. As noted in [48], Woo and Lam's approach is based on truth in extensions of arbitrary default theories, which is known, even in the propositional case to be NP-complete, and in the first order case, is worse than undecidable.

Starting from these observations, Jajodia et al. [48] worked on a proposal for a logic-based language that attempted to balance flexibility and expressiveness on the one side, and easy management and performance on the other. The language allows the representation of different policies and protection requirements, while at the same time providing understandable specifications, clear semantics (guaranteeing therefore the behavior of the specifications), and bearable data complexity. Their proposal for a Flexible Authorization Framework (FAF) identifies a polynomial time (in fact quadratic time) data complexity fragment of default logic; thus resulting effectively implementable. The language identifies the following predicates for the specification of authorizations. (Below s, o, and a denote a subject, object, and action term, respectively, where a term is either a constant value in the corresponding domain or a variable ranging over it).

cando(o,s,⟨*sign*⟩a) represents authorizations explicitly inserted by the security administrator. They represent the accesses that the administrator wishes to allow or deny (depending on the sign associated with the action).

dercando(o,s,⟨*sign*⟩a) represents authorizations derived by the system using logical rules of inference (modus ponens plus rules for stratified negation). Logical rules can express hierarchy-based authorization derivation (e.g., propagation of authorizations from groups to their members) as well as different implication relationships that may need to be represented.

do(o,s,⟨*sign*⟩a) definitely represents the accesses that must be granted or denied. Intuitively, `do` enforces the conflict resolution and access decision policies, that is, it decides whether to grant or deny the access possibly solving existing conflicts and enforcing default decisions (in the case where no authorization has been specified for an access).

done(o,s,r,a,t) keeps the history of the accesses executed. A fact of the form `done(o,s,r,a,t)` indicates that `s` operating in role `r` executed action `a` on object `o` at time `t`.

Stratum	Predicate	Rules defining predicate
0	`hie-predicates`	base relations.
	`rel-predicates`	base relations.
	`done`	base relation.
1	`cando`	body may contain `done`, `hie-` and `rel-`literals.
2	`dercando`	body may contain `cando`, `dercando`, `done`, `hie-`, and `rel-` literals. Occurrences of `dercando` literals must be positive.
3	`do`	in the case when head is of the form $\texttt{do}(\_, \_, +a)$ body may contain `cando`, `dercando`, `done`, `hie-` and `rel-` literals.
4	`do`	in the case when head is of the form $\texttt{do}(o, s, -a)$ body contains just one literal $\neg\texttt{do}(o, s, +a)$.
5	`error`	body may contain `do`, `cando`, `dercando`, `done`, `hie-`, and `rel-` literals.

Fig. 22. Rule composition and stratification of the proposal in [48]

8.2 Composition of access control policies

In many real world situations, access control needs to combine restrictions independently stated that should be enforced as one, while retaining their independence and administrative autonomy. For instance, the global policy of a large organization can be the combination of the policies of its different departments and divisions as well as of externally imposed constraints (e.g., privacy regulations); each of these policies should be taken into account while remaining independent and autonomously managed. Another example is represented by the emerging dynamic coalition scenarios where different parties, coming together for a common goal for a limited time, need to merge their security requirements in a controlled way while retaining their autonomy. Since existing frameworks assume a single monolithic specification of the entire access control policy, the situations above would require translating and merging the different component policies into a single "program" in the adopted access control language. While existing languages are flexible enough to obtain the desired combined behavior, this method has several drawbacks. First, the translation process is far from being trivial; it must be done very carefully to avoid undesirable side effects due to interference between the component policies. Interference may result in the combined specifications not reflecting correctly the intended restrictions. Second, after translation it is not possible anymore to operate on the individual components and maintain them autonomously. Third, existing approaches cannot take into account incomplete policies, where some components are not (completely) known a priori (e.g., when somebody else is to provide that component). Starting from these observations, Bonatti et al. [20] make the point for the need of a policy composition framework by which different component policies can be

different component, provides a convenient way for reasoning about policies at different levels of abstractions. Also, it allows for the support of heterogeneous policies and policies that are unknown a priori and can only be queried at access control time.

8.3 Certificate-based access control

Today's Globally Internetworked Infrastructure connects remote parties through the use of large scale networks, such as the World Wide Web. Execution of activities at various levels is based on the use of remote resources and services, and on the interaction between different, remotely located, parties that may know little about each other. In such a scenario, traditional assumptions for establishing and enforcing access control regulations do not hold anymore. For instance, a server may receive requests not just from the local community of users, but also from remote, previously unknown users. The server may not be able to authenticate these users or to specify authorizations for them (with respect to their identity). The traditional separation between *authentication* and *access control* cannot be applied in this context, and alternative access control solutions should be devised. A possible solution to this problem is represented by the use of digital certificates (or credentials), representing statements certified by given entities (e.g., certification authorities), which can be used to establish properties of their holder (such as identity, accreditation, or authorizations) [39]. Trust-management systems (e.g., PolicyMaker [18], Keynote [17], REFEREE [24], and DL [57]) use credentials to describe specific delegation of trusts among keys and to bind public keys to authorizations. They therefore depart from the traditional separation between authentication and authorizations by granting authorizations directly to keys (bypassing identities). Trust management systems provide an interesting framework for reasoning about trust between unknown parties; however, assigning authorizations to keys, may result limiting and make authorization specifications difficult to manage. A promising direction exploiting digital certificates to regulate access control is represented by new authorization models making the access decision of whether or not a party may execute an access dependent on properties that the party may have, and can prove by presenting one or more certificates (authorization certificates in [18] being a specific kind of them). Besides a more complex authorization language and model, there is however a further complication arising in this new scenario, due to the fact that the access control paradigm is changing. On the one side, the server may not have all the information it needs in order to decide whether or not an access should be granted (and exploits certificates to take the decision). On the other side, however, the requestor may not know which certificates she needs to present to a (possibly just encountered) server in order to get access. Therefore, the server itself should, upon reception of the request, return the user with the information of what she should do (if possible) to get access. In other words the system cannot simply return a "yes/no" access decision anymore. Rather, it should return the information of the requisites that it requires be satisfied for the access to be allowed. The certificates mentioned above are

in [88, 93] investigating trust negotiation issues and strategies that a party can apply to select credentials to submit to the opponent party in a negotiation. In particular, [88] distinguishes between *eager* and *parsimonious* credential release strategies. Parties applying the first strategy turn over all their credentials if the release policy for them is satisfied, without waiting for the credentials to be requested. Parsimonious parties only release credentials upon explicit request by the server (avoiding unnecessary releases). Yu et al. [93] present a prudent negotiation strategy to the goal of establishing trust among parties, while avoiding disclosure of irrelevant credentials.

A credential-based access control is also presented by Bonatti and Samarati in [21]. They propose a uniform framework for regulating service access and information disclosure in an open, distributed network system like the Web. Like in previous proposals, access regulations are specified as logical rules, where some predicates are explicitly identified. Besides credentials, the proposal also allows to reason about declarations (i.e., unsigned statements) and user-profiles that the server can maintain and exploit for taking the access decision. Communication of requisites to be satisfied by the requestor is based on a filtering and renaming process applied on the server's policy, which exploits partial evaluation techniques in logic programs. The filtering process allows the server to communicate to the client the requisites for an access, without disclosing possible sensitive information on which the access decision is taken. The proposal allows also clients to control the release of their credentials, possibly making counter-requests to the server, and releasing certain credentials only if their counter-requests are satisfied (see Figure 23). Client-server interplay is limited to two interactions to allow clients to apply a parsimonious strategy (i.e., minimizing the set of information and credentials released) when deciding which set credentials/declarations release among possible alternative choices they may have.

While all these approaches assume access control rules to be expressed in logic form, often the people specifying the security policies are unfamiliar with logic based languages. An interesting aspect to be investigated concerns the definition of a language for expressing and exchanging policies based on a high level formulation that, while powerful, can be easily interchangeable and both human and machine readable. Insights in this respect can be taken from recent proposals expressing access control policies as XML documents [26, 27].

All the proposals above open new interesting directions in the access control area.

References

1. M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15:706–734, 1993.
2. R. Ahad, J. David, S. Gower, P. Lyngbaek, A. Marynowski, and E. Onuebge. Supporting access control in an object-oriented database language. In *Proc. of*

22. D.F.C. Brewer and M.J. Nash. The Chinese Wall security policy. In *Proc. IEEE Symposium on Security and Privacy*, pages 215–228, Oakland, CA, 1989.
23. S. Castano, M.G. Fugini, G. Martella, and P. Samarati. *Database Security.* Addison-Wesley, 1995.
24. Y-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust management for Web applications. *Computer Networks and ISDN Systems*, 29(8–13):953–964, 1997.
25. D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings IEEE Computer Society Symposium on Security and Privacy*, pages 184–194, Oakland, CA, May 1987.
26. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Design and implementation of an access control processor for XML documents. *Computer Networks*, 33(1–6):59–75, June 2000.
27. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Fine grained access control for SOAP e-services. In *Tenth International World Wide Web Conference*, Hong Kong, China, May 2001.
28. Database language SQL – part 2: Foundation (SQL/foundation). ISO International Standard, ISo/IEC 9075:1999, 1999.
29. C.J. Date. *An Introduction to Database Systems.* Addison-Wesley, 6th edition, 1995.
30. S. Dawson, S. De Capitani di Vimercati, P. Lincoln, and P. Samarati. Minimal data upgrading to prevent inference and association attacks. In *Proc. of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, Philadelphia, CA, 1999.
31. D.E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.
32. D.E. Denning. *Cryptography and Data Security.* Addison-Wesley, Reading, MA, 1982.
33. D.E. Denning. Commutative filters for reducing inference threats in multilevel database systems. In *Proc. of the 1985 IEEE Symposium on Security and Privacy*, pages 134–146, April 1985.
34. S. De Capitani di Vimercati, P. Samarati, and S. Jajodia. Hardware and software data security. In *Encyclopedia of Life Support Systems.* EOLSS publishers, 2001. To appear.
35. E.B. Fernandez, E. Gudes, and H. Song. A model for evaluation and administration of security in object-oriented databases. *IEEE Transaction on Knowledge and Data Engineering*, 6(2):275–292, 1994.
36. D. Ferraiolo and R. Kuhn. Role-based access controls. In *Proc. of the 15th NIST-NCSC Naional Computer Security Conference*, pages 554–563, Baltimore, MD, October 1992.
37. R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9), September 1997.
38. T.D. Garvey and T.F. Lunt. Cover stories for database security. In C.E. Landwehr and S. Jajodia, editors, *Database Security, V: Status and Prospects*, North-Holland, 1992. Elsevier Science Publishers.
39. B. Gladman, C. Ellison, and N. Bohm. Digital signatures, certificates and electronic commerce. http://jya.com/bg/digsig.pdf.
40. J.A Goguen and J. Meseguer. Unwinding and inference control. In *Proc. of the 1984 Symposium on Research in Security and Privacy*, pages 75–86, 1984.

60. T.F. Lunt. Polyinstantiation: an inevitable part of a multilevel world. In *Proc. Of the IEEE Workshop on computer Security Foundations*, pages 236–238, Franconia, New Hampshire, June 1991.
61. T.F. Lunt, D.E. Denning, R.R. Schell, M. Heckman, and W.R. Shockley. The SeaView security model. *IEEE Transactions on Software Engineering*, 16(6):593–607, June 1990.
62. C.J. McCollum, J.R. Messing, and L. Notargiacomo. Beyond the pale of MAC and DAC - Defining new forms of access control. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 190–200, Oakland, CA, 1990.
63. J. McLean. The specification and modeling of computer security. *Computer*, 23(1):9–16, January 1990.
64. J. McLean. Security models. In *Encyclopedia of Software Engineering*. Wiley Press, 1994.
65. Communication of the ACM. Special issue on internet privacy. *CACM*, February 1999.
66. Oracle Corporation, Redwood City, CA. *Trusted Oracle7 Server Administration Guide, Version 7.0*, January 1993.
67. S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, 3(2):85–106, 2000.
68. W.R. Polk and L.E. Bassham. Security issues in the database language SQL. Technical Report NIST special publication 800-8, Institute of Standards and Technology, 1993.
69. X. Qian and T.F. Lunt. A MAC policy framework for multilevel relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):1–14, February 1996.
70. F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM TODS*, 16(1):89–131, March 1991.
71. J. Richardson, P. Schwarz, and L. Cabrera. CACL: Efficient fine-grained protection for objects. In *Proceedings of OOPSLA*, 1992.
72. M. Roscheisen and T. Winograd. A communication agreement framework for access/action control. In *Proc. of 1996 IEEE Symposium on Security and Privacy*, pages 154–163, Oakland, CA, May 1996.
73. P. Samarati and S. Jajodia. Data security. In J.G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*. John Wiley & Sons, 1999.
74. R. Sandhu. On five definitions of data integrity. In *Proc. of the IFIP WG 11.3 Workshop on Database Security*, Lake Guntersville, Alabama, September 1993.
75. R. Sandhu and F. Chen. The multilevel relational (MLR) data model. *ACM Transactions on Information and System Security (TISSEC)*, 2000.
76. R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: Towards a unified standard. In *Proc. of the fifth ACM Workshop on Role-based Access Control*, pages 47–63, Berlin Germany, July 2000.
77. R. Sandhu and Q. Munawer. The ARBAC99 model for administration of roles. In *Proc. of the 15th Annual Computer Security Applications Conference*, Phoenix, Arizona, December 1999.
78. R. Sandhu and P. Samarati. Authentication, access control and intrusion detection. In A. Tucker, editor, *CRC Handbook of Computer Science and Engineering*, pages 1929–1948. CRC Press Inc., 1997.
79. Ravi S. Sandhu. Transaction control expressions for separation of duties. In *Fourth Annual Computer Security Application Conference*, pages 282–286, Orlando, FL, December 1988.

Attacking Malicious Code: A Report to the Infosec Research Council

The accelerating trends of interconnectedness, complexity, and extensibility are aggravating the already-serious threat posed by malicious code. To combat malicious code, these authors argue for creating sound policy about software behavior and enforcing that policy through technological means.

Gary McGraw, Cigital
Greg Morrisett, Cornell University

In October of 1999, the Infosec Research Council created a Science and Technology Study Group focused on malicious code. The Malicious Code ISTSG is charged with developing a national research agenda to address the accelerating threat from malicious code. The study is intended to identify promising new approaches to dealing with the problems posed by malicious code. In this report, we discuss important trends that are making malicious code an increasingly serious problem. We then survey existing techniques for preventing attacks, pointing out their limitations, and discuss some promising new approaches that might address these limitations.

This report is a byproduct of two meetings of Study Group members and their invited guests. Although this report was written by two of the study group members, we believe it represents an accurate distillation of the ideas and insights of all the participants.

What is Malicious Code?

Malicious code is any code added, changed, or removed from a software system to intentionally cause harm or subvert the system's intended function. Although the problem of malicious code has a long history, a number of recent, widely publicized attacks and certain economic trends suggest that malicious code is rapidly becoming a critical problem for industry, government, and individuals.

Traditional examples of malicious code include viruses, worms, Trojan Horses, and attack scripts, while more modern examples include Java attack applets and dangerous ActiveX controls:

- Viruses are pieces of malicious code that attach to host programs and propagate when an infected program executes.
- Worms are particular to networked

0740-7459/00/$10.00 © 2000 IEEE

Extensible systems, including computers, are particularly susceptible to the malicious functionality problem.

computers. Instead of attaching themselves to a host program, worms carry out programmed attacks to jump from machine to machine across the network.

- Trojan Horses, like viruses, hide malicious intent inside a host program that appears to do something useful (such as a program that captures passwords by masquerading as the login daemon).
- Attack scripts are programs written by experts that exploit security weaknesses, usually across the network, to carry out an attack. Attack scripts exploiting buffer overflows by "smashing the stack" are the most commonly encountered variety.
- Java attack applets are programs embedded in Web pages that achieve foothold through a Web browser.
- Dangerous ActiveX controls are program components that allow a malicious code fragment to control applications or the operating system.

Recently, the distinctions between malicious code categories have been bleeding together, making classification difficult. Table 1 provides some concrete examples of malicious code. Recent versions of malicious code are really amalgamations of different categories.

A Growing Problem

Complex devices, by their very nature, introduce the risk that malicious functionality can be added (either during creation or afterwards) that extends the original device past its primary intended design. As an unfortunate side effect, inherent complexity lets malicious subsystems remain invisible to unsuspecting users until it is too late. Some of the earliest malicious functionality, for example, was associated with complicated copy machines. Extensible systems, including computers, are particularly susceptible to the malicious functionality problem. When extending a system is as easy as writing and installing a program, the risk of intentional introduction of malicious behavior increases drastically.

Any computing system is susceptible to malicious code. Rogue programmers can modify systems software that is initially installed on the machine. Users might unwittingly propagate a virus by installing new programs or software updates from a CDROM. In a multi-user system, a hostile user might install a Trojan Horse to collect other users' passwords. These attack vectors have been well known since the dawn of computing, so why is malicious code a bigger problem now than in the past? We argue that a small number of trends have a large influence on the recent widespread propagation of malicious code.

Networks Are Everywhere

The growing connectivity of computers through the Internet has increased both the number of attack vectors and the ease with which an attack can be made. More and more computers, ranging from home PCs to systems that control critical infrastructures (such as the power grid), are being connected to the Internet. Furthermore, people, businesses, and governments are increasingly dependent upon network-enabled communication such as e-mail or Web pages provided by information systems. Unfortunately, as these systems are connected to the Internet, they become vulnerable to attacks from distant sources. Put simply, an attacker no longer needs physical access to a system to install or propagate malicious code.

Because access through a network does not require human intervention, launching automated attacks from the comfort of your living room is relatively easy. Indeed, the recent denial-of-service attacks in February of 2000 took advantage of a number of (previously compromised) hosts to flood popular e-commerce Web sites with bogus requests automatically. The ubiquity of networking means that there are more systems to attack, more attacks, and greater risks from malicious code than in the past.

System Complexity Is Rising

A second trend that has enabled widespread propagation of malicious code is the size and complexity of modern information systems. A desktop system running Windows/NT and associated applications depends upon the proper functioning of the kernel as well as the applications to ensure that malicious code cannot corrupt the system. However, NT itself consists of tens of millions of lines of code, and applications are becoming equally, if not more, complex. When systems become this large, bugs cannot be avoided. Exacerbating this problem is the use of unsafe programming languages

Table 1

Examples of malicious code.

Malicious code	Date	Category	Explanation
Love Bug	2000	Mobile code virus	The fastest spreading virus of all time used VB script and Microsoft Outlook [illegible] to propagate. Caused an estimated $10 billion in damages. [illegible]
Trinoo (and other dDoS scripts)	2000	Remote-control attack script	The highly publicized denial-of-service attacks of February 2000 [illegible] planted agent programs. [illegible]
Melissa	1999	Mobile code virus	The second fastest spreading virus of all time [illegible]
ExploreZip	1999	Mobile code worm	An e-mail borne worm that exploited problems in Microsoft [illegible]
Happy99	1999	Virus	A widespread virus infecting Microsoft PCs [illegible]
CIH	1998	Virus	A particularly dangerous virus that affects BIOS in PCs. [illegible]
Back Orifice	1998	Offensive code	Remote-control program installed on Windows computers [illegible]
Attack scripts	[illegible]	Offensive code	Crackers called "script kiddies" download malicious code from the [illegible] number of targets. [illegible] Most common attack: buffer overflow. [illegible]
ActiveX (scripting)	1997	Mobile code	Decried by security professionals, Microsoft's ActiveX system introduce[illegible] relying on user's discretion and judgment [illegible] digital signature [illegible]
Java Attack Applets	1996–2000	Mobile code	Attack applets placed on Web sites take advantage of flaws in the Java [illegible] out attacks. 19 known attacks. [illegible]
Morris Worm	1988	Worm	Released in 1988 by Robert Morris, Jr. this program affected around 6,000 computers (around 10% of the Internet at the time).
Thompson's compiler trick	1984	Trojan Horse	Ken Thompson introduced a Trojan Horse in a C compiler that compile[illegible] programs ("Reflections on Trusting Trust," *Comm. ACM*, Vol. 27, No. 8, Aug. 1984).

(such as C or C++) that do not protect against simple kinds of attacks, such as buffer overflows. However, even if the systems and applications code were bug free, improper configuration by retailers, administrators, or users can open the door to malicious code. In addition to providing more avenues for attack, complex systems make it easier to hide or mask malicious code. In theory, we could analyze and prove that a small program was free of malicious code, but this task is impossible for even the simplest desktop systems today, much less the enterprise-wide systems used by businesses or governments.

Systems Are Easily Extensible

A third trend enabling malicious code is the degree to which systems have become extensible. An extensible host accepts updates or extensions, sometimes referred to as *mobile code*, so that the system's functionality can be evolved in an incremental fashion. For example, the plug-in architecture of Web browsers makes it easy to install viewer extensions for new document types as needed. Today's operating systems support extensibility through dynamically loadable device drivers and modules. Today's applications, such as word processors, e-mail clients, spreadsheets, and Web browsers, support extensibility through scripting, controls, components, and applets. From an economic standpoint, extensible systems are attractive because they provide flexible interfaces that can be adapted through new components. In today's marketplace, it is crucial that software be deployed as rapidly as possible to gain market share. Yet the marketplace also demands that applications provide new features with each release. An extensible architecture makes it easy to satisfy both demands by letting companies ship the base application code early and later ship feature extensions as needed.

Unfortunately, the very nature of extensible systems makes it hard to prevent malicious code from slipping in as an unwanted extension. For example, the Melissa virus took advantage of the scripting extensions of Microsoft's Outlook e-mail client to propagate itself. The virus was coded as a script contained in what appeared to users as an innocuous mail message. When users opened the message, the script executed, proceeded to obtain email addresses from the user's contacts database, and then sent copies of itself to those addresses. The infamous Love Bug worked very similarly, also taking advantage of Outlook's scripting capabilities.

Defense against Malicious Code

Creating malicious code is not hard. In

To make matters worse, our traditional tools for ensuring the security and integrity of hosts have not kept pace with the ever-changing suite of applications.

fact, it is as simple as writing a program or downloading and configuring a set of easily customized components. It is becoming increasingly easy to hide ill-intentioned code inside otherwise innocuous objects, including Web pages and e-mail messages. This makes detecting and stopping malicious code before it can do any damage extremely hard.

To make matters worse, our traditional tools for ensuring the security and integrity of hosts have not kept pace with the ever-changing suite of applications. For example, traditional security mechanisms for access control reside within an operating system kernel and protect relatively primitive objects (such as files); but increasingly, attacks such as the Melissa virus happen at the application level where the kernel has no opportunity to intervene.

A useful analogy is to think of today's computer and network security mechanisms like the walls, moats, and drawbridges of medieval times. At one point, these mechanisms were effective for defending our computing castles against isolated attacks, mounted on horseback. But the defenses have not kept pace with the attacks. Today, attackers have access to airplanes and laser-guided bombs that can easily bypass our antiquated defenses. In fact, attackers rarely need sophisticated equipment: because our kingdoms are really composed of hundreds of interconnected castles, attackers can easily move from site to site, finding places where we have left the drawbridge down. It is time to develop some new defenses.

In general, when a computational agent arrives at a host, there are four approaches that the host can take to protect itself:

- ***Analyze*** the code and reject it if there is the potential that executing it will cause harm.
- ***Rewrite*** the code before executing it so that it can do no harm.
- ***Monitor*** the code while its executing and stop it before it does harm, or
- ***Audit*** the code during executing and take policing action if it did some harm.

Code analysis includes simple techniques, such as scanning a file and rejecting it if contains any known virus, as well as more sophisticated techniques, including dataflow analysis, which can sometimes discover previously unseen malicious code. Analysis can also help locate security-related bugs (such as potential buffer overflow conditions) that malicious code can use to gain a foothold in a system. But analyses are necessarily limited, because determining if code will misbehave is as hard as the halting problem. Consequently, any analysis will either be too conservative (and reject some perfectly good code) or too permissive (and let some bad code in) or more likely, both. Furthermore, software engineers working on their own systems often neglect to apply any bug-finding analyses. Automated tools such as the open source security scanner ITS4 (see www.rstcorp.com/its4) and more sophisticated tools incorporating dataflow analysis can be effective for finding bugs.[1,2] In addition, primitive static analysis, such as looking for particular patterns of system calls in an executable, has been incorporated into some commercially available security products.

Code rewriting is a less pervasive approach to the problem, but might become more important (see the next section). With this approach, a rewriting tool inserts extra code to perform dynamic checks that ensure bad things cannot happen. For example, a Java compiler inserts code to check that each array index is in bounds—if not, the code throws an exception, thereby avoiding the common class of buffer overflow attacks. Rewriting can be carried out either at the application code level, or below that in subsystem functionality made available through APIs, or even at the binary level.

Monitoring programs, using a reference monitor, is the traditional approach used to ensure programs do not do anything bad. For instance, an operating system uses the page-translation hardware to monitor the set of addresses that an application attempts to read, write, or execute. If the application attempts to access memory outside of its address space, the kernel takes action (such as by signaling a segmentation fault.) A more recent example of an online reference monitor is the Java Virtual Machine interpreter. The interpreter monitors execution of applets and mediates access to system calls by examining the execution stack to determine who is issuing the system call request.[3] In this case, stack inspection serves as a policy enforcement mechanism.

If malicious code does damage, recovery is only possible if the damage can be prop-

erly assessed and addressed. Creating an audit trail that captures program behavior is an essential step. Several program-auditing tools are commercially available.

Each of the basic approaches—analysis, rewriting, monitoring, and auditing—has its strengths and weaknesses, but fortunately, these approaches are not mutually exclusive and can be used in concert. Of course, to employ any of them, we must first identify what could be "harmful" to a host. Like any other computing task, we must turn the vague idea of "harm" into a concrete, detailed specification—a security policy—so that it can be enforced by some automated security architecture. Therein lies our greatest danger, for as we create the policy, we are likely to abstract or forget relevant details of the system. An attacker will turn to these details first, stepping outside our policy model to circumvent the safeguards.

Stick to Your Principles

To protect against this common failing, it is important to follow well-established security principles when designing security policies. One of the most important principles, first stated by Jerome Saltzer and Michael Shroeder in 1975,[4] is the *Principle of Least Privilege:* a component should be given the minimum access necessary to accomplish its intended task. For example, we shouldn't give a program access to all files in a system but rather, only those files that the program needs to get its job done. This prevents the program from either accidentally or maliciously deleting or corrupting most files. Obviously, the fewer files that the program can access, the less the potential damage. Stated simply, tighter constraints on a program lead to better security.

Another important security principle is the *Principle of Minimum Trusted Computing Base.* The trusted computing base (TCB) is the set of hardware and software components that make up our security enforcement mechanisms. The Principle of Minimum TCB states that, in general, the best way to assure that your system is secure is to keep your TCB small and simple. Even in the mid 1970's, operating system kernels were thought to be too large to be trusted. Those systems now seem small and tightly structured compared to today's widely used kernels composed of millions of lines of code.

Current Defenses

We now turn to examples of currently deployed defenses for malicious code, focusing on their relative pros and cons. Unfortunately, the comparison shows that the pros are outweighed by the cons, largely because of a violation of the Least Privilege and Minimal TCB principles.

OS-Based Reference Monitors

Historically, mechanisms for security policy enforcement have been provided by the computer hardware and operating system. Address translation hardware, distinct supervisor- and user-modes, timer interrupts, and system calls for invoking a trusted software base serve in combination to enforce limited forms of availability, fault containment, and authorization properties.

To a large degree, these mechanisms have proven effective for protecting operating system resources (such as files or devices) from unauthorized access by humans or malicious code. But the mechanisms work with a fixed system-call interface and a fixed vocabulary of principals, objects, and operations for policies. Only by incurring significant cost and usability penalties can that vocabulary be expanded. It rarely is. Currently, most desktop machines are configured as single-user, so applications have complete access to the machine resources.

Scanning for Known Malicious Code

In the days before networking was rampant, malicious code mostly used the "sneaker net" as its vector. Viruses spread from machine to machine by humans carrying floppy disks with infected programs on them. Perhaps the built-in limitations in the vector kept the number of viruses small. In any case, the limited number of viruses combined with the inefficiencies in the communication vector made possible the strategy *of blacklisting.*

Blacklisting, a strategy used by most commercial antivirus products, matches programs against a database of known virus signatures (such as code fragments). If a match is found, the program is disabled. Today, commercial products scan not only binary programs, but also email messages, Web pages, or documents looking for viruses in the form of scripts. This approach's limitations are obvious. Unknown malicious code will easily get by the simple defenses to carry

In the days before networking was rampant, malicious code mostly used the "sneaker net" as its vector. Viruses spread from machine to machine by humans carrying floppy disks with infected programs on them.

Type systems, like software-based reference monitors, go beyond operating systems in that they can be used to enforce a wider class of application-specific access policies.

out its dirty work. Until vendors can contain a new virus and add a signature entry to the database, it can run rampant. Recall both the Melissa virus and the Love Bug. Another limitation of the approach is that it does not scale well; each file must be scanned against an ever growing list of viruses.

Clearly, blacklisting by itself does not provide adequate security. It is too easy to make trivial changes to malicious code (a process that can be automated in the code itself) to thwart almost every black listing scheme. Nevertheless, black listing is cheap to implement and is thus worthwhile even if it only stops the occasional naïve attack.

Code Signing

Code signing is an approach for authenticating code based on public-key cryptography and digital signatures. The digital signature lets a user determine which particular key the code was signed with and ensure (with high probability) that the code has not been tampered with since it was signed.

Unfortunately, most people assume that digital signatures imply a lot more than they really do. In particular, people typically assume that the signed code was signed by the owner of the key, that the owner of the key wrote the code, that the code is good, and that the code may be safely used in any context. But these assumptions are often not true!

For instance, if a key is stolen, anyone can use it to sign any piece of code—including malicious code. As another example, the developer might consider the code to be "good" and thus sign it, even if the code contains a Trojan horse or virus. Finally, what developers or retailers consider to be good might not be good for the user: A component that sends back information to the home office may seem useful to a vendor, but will probably be considered a violation of privacy by the user.

Thus, while code signing is a useful technology, it suffers from some real limitations not the least of which is poor understanding of what a digital signature really means. Furthermore, the adoption of code signing has been hampered by the lack of a Public Key Infrastructure. Very few PKI installations have been deployed, and those that have do not begin to approach Internet scale. Without a solid PKI, code signing will not become common.

Promising New Defenses

Now we'll discuss some promising technologies, identified by the study group, that are emerging from research labs.

Software-Based Reference Monitors

Robert Wahbe and his colleagues suggested *software-based fault isolation* as an alternative to the traditional hardware-based mechanisms used to ensure memory safety.[5] Their goal was to reduce the overhead of cross-domain procedure calls and providing a more flexible memory-safety mode. Their basic idea is to rewrite binary code by inserting checks on each memory access and each control transfer to ensure that those accesses are valid. Fred Schneider generalized the SFI idea to in-lined reference monitors.[6] With the IRM approach, a security policy is specified in a declarative language, and a general-purpose tool rewrites code, inserting extra checks and state that serve to enforce the policy. In principle, any security policy that is a safety property can be enforced, so the approach is quite powerful. For example, it can enforce any discretionary access control policy. The approach is also practical: Prototypes have been built at both Cornell and MIT.[7–9] One of the Cornell prototypes, PSLang/PoET, works for the Java Virtual Machine language and gives competitive performance for the implementation of Java's stack inspection security policy.

Type-Safe Languages

Type-safe programming languages, such as Java, Scheme, or ML, ensure that operations are only applied to values of the appropriate type. Type systems that support type abstraction let programmers specify new, abstract types and signatures for operations that prevent unauthorized code from applying the wrong operations to the wrong values. In this respect, type systems, like software-based reference monitors, go beyond operating systems in that they can be used to enforce a wider class of application-specific access policies. Static type systems also enable offline enforcement through static type checking instead of each time a particular operation is performed. This lets the type checker enforce certain policies that are difficult with online techniques. For ex-

Table 2

Examples of malicious code understood in our policy-based framework.

Bad policy	Examples	Incorrect policy enforcement	Examples
Context misunderstood	Scripting languages	Mechanism too weak	Privacy in smart cards
Overly restrictive	Changing passwords too frequently	Mechanism is buggy	Buffer overflows in Java
Noncomprehensive	Executable content in e-mail	Mechanism is misconfigured	Sendmail debug mode

ample, Andrew Myers' Jflow extends the Java type system to enforce the policy that high-security data should never be leaked.[10] Current research in type systems is aimed at eliminating more run-time checks (such as array bounds checks[11]) or type-checking machine code[12].

Proof-Carrying Code

Proof-carrying code (PCC), a concept introduced by George Necula and Peter Lee,[13] is a promising approach for gaining high assurance of security in systems. The basic idea is to require any untrusted code to come equipped with an explicit, machine-checkable *proof* that the code respects a given security policy. Before executing the code, we simply verify that the proof is valid with respect to both the code and the policy. Because proof checkers can be quite simple (Necula's is about six pages of C code), it is easier to ensure that they are correct. And in principle, PCC can enforce any security policy—not just type safety—as long as the code producer can construct a proof. Necula and Lee have shown that such proofs can be constructed automatically for standard type-safety policies, if a compiler for a type-safe programming language generates the code. Unfortunately, going beyond standard notions of type safety cannot be performed automatically without either restricting the code or requiring human intervention. It is unlikely that programmers will construct explicit proofs. Thus an active area of research is how to integrate compilers and modern theorem provers to produce PCC.

Policy as Achilles' Heel

Thus far, we have focused on technology solutions to the malicious code problem. To be sure, technology can be of service; but there is another critical aspect of the problem that remains to be addressed—the problem of policy.

In current forms, extensible systems do little to determine how a system will behave when extended in certain ways or, put another way, what a particular piece of code can and cannot do. In fact, today's computers are hyper-malleable and overly complicated. This greatly increases the malicious code risk. In the end, determining whether something malicious is happening requires first defining some policy to enforce.

When Policy Breaks Down

Clearly, the notion of policy is deeply intertwined with the concept of malicious code. Understood in terms of policy, the root causes of malicious code fall into two basic categories: bad policy and incorrectly enforced policy.

Bad policy allows malicious code to do something malicious because policy does not forbid it. Even if policy is perfectly enforced by technology, the policy itself must be well formed. Subcategories of bad policy include:

- misunderstandings of context, whereby policy makes no sense in the context where it was applied;
- over restriction, whereby the policy prevents useful work when it is enforced; or
- noncomprehensiveness, whereby policy fails to cover some situation or exists at the wrong level of abstraction.

Incorrect policy enforcement allows code to do something malicious even if it is correctly forbidden by policy. This situation arises when either

- the enforcement mechanism is too weak to implement the desired policy;
- there are bugs in the implementation of the enforcement mechanism; or
- the enforcement mechanism is misconfigured.

Table 2 provides examples of malicious code understood in our policy-based framework.

As an example of context misunderstanding, consider the role of scripting lan-

Meet the Authors

Gary McGraw is the vice president of Corporate Technology at Cigital (formerly known as Reliable Software Technologies) where he pursues research in software security while leading the Software Security Group. He has served as principal investigator on grants from AFRL, DARPA, NSF, and NIST's Advanced Technology Program. He chairs the National Infosec Research Council's Malicious Code Infosec Science and Technology Study Group. He coauthored both *Java Security* (Wiley, 1996) and *Securing Java* (Wiley, 1999), and is currently writing a book entitled *Building Secure Software* (Addison-Wesley, 2001). Contact him at Cigital, 21351 Ridgetop Circle, Ste. 400, Dulles, VA 20166; gem@rstcorp.com.

Greg Morrisett is an assistant professor in the Computer Science department at Cornell University. He is a Sloan Research Fellow, recipient of an NSF Career award, member of the IFIP Working Group 2.8 (Functional Programming), editor for the *Journal of Functional Programming*, and an associate editor for *ACM Transactions on Programming Languages and Systems*. His research interests include programming language-based security and type systems. Contact him at the Dept. Computer Science, 4133 Upson Hall, Cornell Univ., Ithaca, NY 15213; jgm@cs.cornell.edu.

guages such as Visual Basic. Such languages can be can be extremely useful and perfectly safe in some contexts. But in other contexts, scripts can be extremely dangerous. For instance, there is rarely a need for scripts or macros to be run when displaying a document, yet this functionality is exactly what the Melissa virus exploited.

An example of a policy that is too restrictive is one in which users are required to pick new passwords at small intervals. Under such a policy, people often forget their current password. To avoid this, they may write down their password in an insecure place, making it easier for an attacker to steal it.

Most security policies fail to be comprehensive, simply because designers cannot think of all possible attacks. For instance, in the early days of the Internet, there was no need for an e-mail security policy, because mail readers did not interpret messages. Today, messages can contain attachments or scripts that are automatically executed by readers.

Many desirable security policies just aren't achievable or practical. For instance, stringent policies have been formulated for smart cards to prevent disclosure of private or secret information, such as health records or crypto keys. But hiding information is a tricky business and just about any enforcement mechanism will fail to block all information flow off the card. In the case of smart cards, the designers used clever algorithms and packaging techniques to prevent tampering with the card to learn private information. But they failed to take into account of the power fluctuations across the connection pins—data that can be used to reconstruct private information.[14]

Sometimes an enforcement mechanism is powerful enough to implement the policy, but its implementation has bugs or weaknesses that prevent it from doing so. The classic example of such bugs are the buffer overflow attacks that arise in operating systems and applications.

Finally, sometimes the enforcement mechanism is powerful enough and coded properly, but simply misconfigured. For example, the sendmail program has debugging features that allow a programmer to gain remote access to a machine. During development, this feature was turned on. Unfortunately, the feature remained on when sendmail was deployed and subsequent attacks such as the Morris worm took advantage of the opening.

Addressing the malicious code problem requires the creation of sound policy and its careful enforcement through technology.

The Many Levels of Policy

System administrators and MIS security people think about policy in terms of user groups, firewall rules, and computer use. Security researchers steeped in programming languages think about policy in terms of memory safety and liveness properties. Government policy wonks think about policy in terms of rules and regulations imposed on users and systems. The problem is, all of these ways of thinking about policy are equally valid!

So how are we to set policy to combat malicious code? We believe the key is to focus on defining metalevel policies that system administrators work with naturally in terms of collections of lower-level enforcement mechanisms. This is no trivial undertaking.

Most of the technologies we've explored earlier in this article can serve to enforce particular aspects of software behavior. Some language researchers, for example, consider the issue of enforcing safety properties "solved," at least in theory. Enforcing liveness properties or confidentiality is harder, but fairly clear research agendas exist to address the open issues. Of course, the terms *safety*, *liveness*, and *confidentiality* have technical meanings. Intuitively, a safety property states that a program will never perform a bad action, for some precisely defined notion of "bad." An example of a bad action is overflowing a buffer. A liveness property, on the other hand, states that a program will eventually perform some desired action or set of actions. For example, the property that a program will

eventually release all of the memory that it allocates is a liveness property. Finally, confidentiality is meant to ensure that certain values remain private or secret.

The problem is that low-level properties such as safety and liveness do not align nicely with what most security administrators think of as policy building blocks. Thus an open question is how to express reasonable security policy that can be directly transformed into technology enforcement solutions.

The answer is to understand policy as a layered set of abstractions. Some preliminary work exists (for example Netscape Navigator's approach to policy sets based on expected code behavior), but much work remains to be done.

The malicious code problem will continue to grow as the Internet grows. The constantly accelerating trends of interconnectedness, complexity, and extensibility make addressing the problem more urgent than ever. As extensible information systems become more ubiquitous, moving into everyday devices and playing key roles in life-critical systems, the level of the threat moves out of the technical world and into the real world. We *must* work on this problem.

Our best hope in combating malicious code is creating sound policy about software behavior and enforcing that policy through the use of technology. An emphasis on one or the other alone will do little to help. Any answer will require a set of enforcement technologies that can be directly tied to policy set and understood by nontechnical users.

Our best hope in combating malicious code is creating sound policy about software behavior and enforcing that policy through the use of technology.

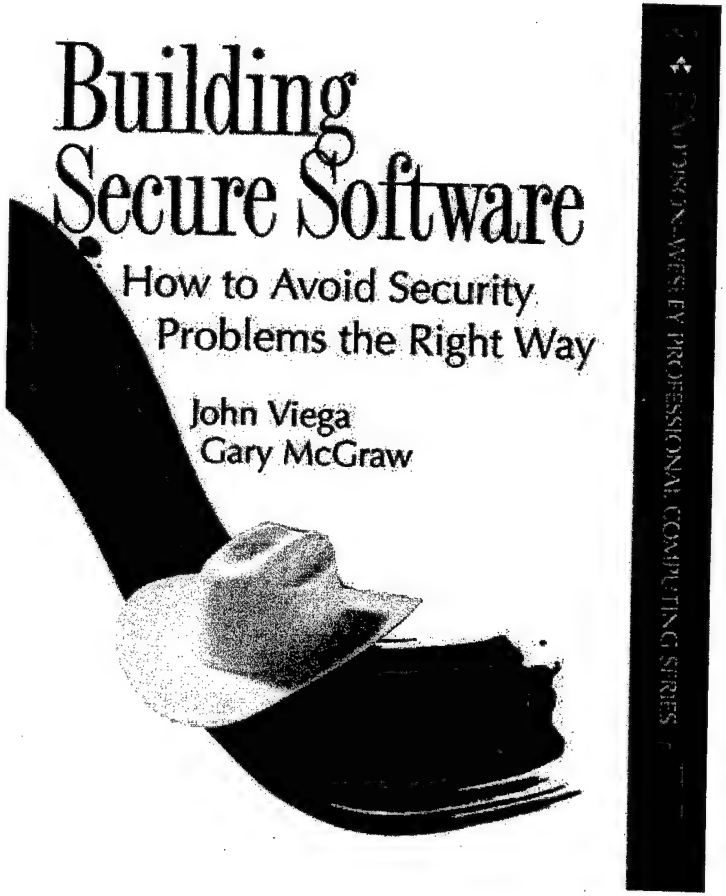
Acknowledgements

Study Group members include Gary McGraw, Cigital, Chair; Avi Rubin, AT&T Research; Ed Felten, Princeton; Peter G. Neumann, SRI; Lee Badger, NAI Labs; Greg Morrisett, Cornell; Tim Teitelbaum, Grammatech; Virgil Gligor, University of Maryland; Tom Markham, Secure Computing; Jay Lepreau, University of Utah; Bob Balzer, ISI; Joshua Haines, Lincoln Labs; Roger Thompson, ICSA.net; Bob Clemons, NSA; Penny Chase, MITRE; Carl Landwehr, Mitretek; Brad Arkin, Reliable Software Technologies; Sami Saydjari, DARPA; Brian Witten, DARPA; and Dave Thompson, Mitretek. Guests who participated in the two day San Antonio workshop include mudge, the l0pht; Crispin Cowen, Wirex; Fred Schneider, Cornell; Peter Lee, CMU; Richard Smith, pharlap; John Rushby, SRI; Dan Wallach, Rice University; Amy Felty, University of Ottawa; and David Evans, University of Virginia.

The workshops on which this report is based were convened under the auspices of the Infosec Research Council (IRC), with members from US Government organizations that sponsor and conduct information security research. Views expressed in the report are those of the authors and may not reflect those of the IRC, its members, or the organizations they represent.

References

1. J. Viega et al., "ITS4: A Static Vulnerability Scanner for C and C++ Code," To appear in *Proc. Ann. Computer Secutiry Applications Conf. 2000*, IEEE Computer Soc. Press, Los Alamitos, Calif: the ITS4 tool is available at www.rstcorp.com/its4.
2. D. Wagner et al., "A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities," *Proc. Network and Distributed Systems Security Symposium (NDSS 2000)*, Internet Soc., Reston, Va., 2000, pp. 3–18.
3. G. McGraw and E. Felten, *Securing Java: Getting Down to Business with Mobile Code*, John Wiley & Sons, New York, 1999; complete Web edition at www.securingjava.com.
4. J.H. Salzter and M.D. Schroeder, "The Protection of Information in Computer Systems," *Proc. IEEE*, IEEE Press, Piscataway, N.J., Vol. 9, No. 63, 1975, pp. 1278–1308.
5. R. Wahbe et al., "Efficient Software-Based Fault Isolation," *Proc. 14th ACM Symp. Operating System Principles (SOSP)*, ACM Press, New York, 1993, pp. 203–216.
6. F. Schneider, "Enforceable Security Policies," *ACM Trans. Information and System Security*, Vol. 2, No. 4, Mar. 2000.
7. U. Erlingsson and F.B. Schneider, "IRM Enforcement of Java Stack Inspection," *IEEE Symp. Security and Privacy*, IEEE Press, Piscataway, N.J., 2000.
8. D. Evans and A. Twyman, "Policy-Directed Code Safety," *Proc. IEEE Symp. Security an Privacy*, IEEE Press, Piscataway, N.J., 1999; see also www.cs.virginia.edu/~evans.
9. U. Erlingsson, U. and F.B. Schneider, "SASI Enforcement of Security Policies: A Retrospective," *Proc. New Security Paradigms Workshop*, ACM Press, New York, 1999, pp. 246–255.
10. A.C. Myers, "JFlow: Practical Mostly Static Information Flow Control," *Proc. 26th ACM Symp. Principles of Programming Languages (POPL)*, ACM Press, New York, 1999, pp. 228–241.
11. H. Xi and F. Pfenning, "Dependent Types in Practical Programming," *Proc. 26th ACM Symp. Principles of Programming Languages (POPL)*, ACM Press, New York, 1999, pp. 214–227.
12. G. Morrisett et al., "From System-F to Typed Assembly Language," *ACM Trans. Programming Languages and Systems*, Vol., 21, No. 3, May 1999, pp. 528–569; www.cs.cornell.edu/talc.
13. G.C. Necula, "Proof-Carrying Code," *Proc. 24th ACM Symp. Principles of Programming Languages (POPL)*, ACM Press, New York, 1997, pp. 106–119; www-nt.cs.Berkeley.edu/home/necula/public_html/pcc.html.
14. P. Kocher, J. Jaffee, and B. Jun, "Differential Power Analysis: Leaking Secrets," *Advances in Cryptology–CRYPTO'99*. . In M. Weiner, ed., Lecture Notes in Computer Science, Vol. 1666, Springer, New York, Aug. 1999, pp. 388-397; www.cryptography.com/dpa/Dpa.pdf.



{final submitted manuscript}

Building Secure Software

How to avoid security problems the right way

By John Viega and Gary McGraw
All comments to viega@list.org and gem@cigital.com

Chapter one: Introduction to Software Security

Building Secure Software will be released in September 2001 by Addison-Wesley. Copies may be ordered in advance from your favorite bookstore.

Chapter 1: Introduction to Software Security

"...any program, no matter how innocuous it seems, can harbor security holes. ... I thus have a firm belief that everything is guilty until proven innocent."
-Steve Bellovin

Computer security is an important topic. As e-commerce blossoms, and the Internet works its way into every nook and cranny of our lives, security and privacy come to play an essential role. Computer security is moving beyond the realm of the technical elite, and is beginning to have a real impact on our everyday lives.

It is no big surprise, then, that security seems to be popping up everywhere, from headline news to TV talk shows. Since the general public doesn't know very much about security, a majority of the words devoted to computer security cover basic technology issues such as what firewalls are, what cryptography is, or which anti-virus product is best. Much of the rest of the computer security coverage centers around the "hot topic of the day," usually involving an out-of-control virus or a malicious attack. Historically, the popular press pays much attention to viruses and denial of service attacks; many people remember hearing about the Anna Kournikova worm, the 'Love Bug', or the Melissa virus ad nauseam. These topics are important, to be sure. Nonetheless, the media generally manages not to get to the heart of the matter when reporting on these subjects. Behind every computer security problem and malicious attack lies a common enemy—bad software.

It's all about the Software

The Internet continues to change the role that software plays in the business world, fundamentally and radically. Software no longer simply supports back offices and home entertainment; instead, software has become the lifeblood of our businesses and has become deeply entwined into our lives. The invisible hand of Internet software enables e-business; automates supply chains; and provides instant, worldwide access to information. At the same time, Internet software is moving into our cars, our televisions, our home security systems and even our toasters.

The biggest problem in computer security today is that many security practitioners don't know what the problem is. Simply put, it's the software! You may have the world's best firewall, but if you let people access an application through the firewall and the code is remotely exploitable, then the firewall will not do you any good (not to mention the fact that the firewall is often a piece of fallible software itself). The same can be said of

cryptography. In fact, 85% of CERT security advisories[1] could not have been prevented with cryptography [Schneider, 1998].

Data lines protected by strong cryptography make poor targets. Attackers like to go after the programs at either end of a secure communications link, since the endpoints are typically easier to compromise. As security professor Gene Spafford puts it, "Using encryption on the Internet is the equivalent of arranging an armored car to deliver credit-card information from someone living in a cardboard box to someone living on a park bench."

Internet-enabled applications, including those developed internally by a business, present the largest category of security risk today. Real attackers compromise software. Of course, software does not need to be Internet-enabled to be at risk; the Internet is just the most obvious avenue of attack in most systems.

This book is about protecting yourself by building secure software. We approach the software security problem as a risk management problem. The fundamental technique is to begin early, know your threats, design for security, and subject your design to thorough objective risk analyses and testing. Our goal is to provide tips and techniques that architects, developers, and managers can use to produce Internet-based code that is as secure as necessary.

A good risk-management approach acknowledges that security is often just a single concern among many, including time to market, cost, flexibility, reusability and ease of use. Organizations must set priorities, and identify the relative costs involved in pursuing each. Sometimes security will not be a high priority.

Some people disagree with a risk management security approach. Many people would like to think of security as a yes-or-no, black-or-white affair, but it's not. You can never prove that any moderately complex system is secure. Often, it's not even worth making a system as secure as possible, because the risk is low and the cost is high. It's much more realistic to think of software security as risk management than as a binary switch that costs a lot to turn on.

Software is at the root of all common computer security problems. If your software misbehaves, a number of diverse sorts of problems can crop up: reliability, availability, safety, and security. The extra twist in the security situation is that a bad guy is actively trying to make your software misbehave. This certainly makes security a tricky proposition.

Malicious hackers don't create security holes, they simply exploit them. Security holes and vulnerabilities, the real root cause of the problem, are the result of bad software design and implementation. Bad guys build exploits (often widely distributed as scripts) that exploit the holes. (By the way, we'll try to refer to bad guys who exploit security

[1] CERT is an organization that studies Internet security vulnerabilities, and occasionally releases security advisories when there are large security problems facing the Internet. See http://www.cert.org.

holes as *malicious hackers* instead of simply *hackers* throughout this book. See the side bar.)

We want to do our part in the war against crummy software by providing lots of information about common mistakes, and good ideas that can help you build more secure code.

Dealing with Widespread Security Failures

We probably don't need to spend too much time convincing you that security holes in software are common, and that you need to watch out for them. Nonetheless, many people do not realize how widespread a problem insecure software really is.

The December 2000 issue of the Industry Standard (a new economy business magazine) featured an article entitled "Asleep at the Wheel" by David Lake. The article emphasized the magnitude of today's security problem using an excellent presentation of data. Using numbers derived from Bugtraq (a mailing list dedicated to reporting security vulnerabilities), David Lake created the figure below, which shows the number of new vulnerabilities reported monthly, from the start of 1998 until the end of September, 2000. According to these data, the number of software holes being reported is growing. These figures suggest that approximately 20 new vulnerabilities in software are made public each week. Some of these vulnerabilities are found in source-available software programs, but many are also found in proprietary code. Similarly, both Unix and Windows programs are well-represented in such vulnerability data. For example, there were over a dozen security problems found and fixed in Microsoft Outlook during the year 2000.

Additionally, "tried and true" software may not be as safe as one might think; many vulnerabilities that have been discovered in software existed for months, years and even decades before discovery, even when the source was available (see Chapter 4).

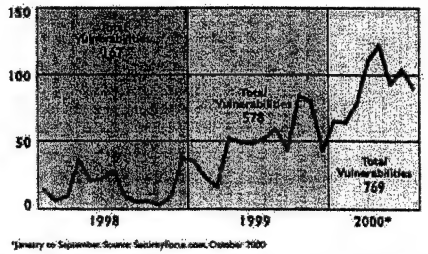


Figure 1: Bugtraq vulnerabilities by month from January 1998 to September 2000.

The consequences of security flaws vary. Consider that the goal of most malicious hackers is to "own" a networked computer (and note that most malicious attackers seem to break into computers simply because they can). Attacks tend to be either "remote" or

"local". In a remote attack, a malicious attacker can break onto a machine that is connected to the same network, usually through some flaw in software. If the software is available through a firewall, then the firewall will be useless. In a local attack, a malicious user can gain additional privileges on a machine, usually administrative privileges. Most security experts agree that once an attacker has a foothold on your machine, it is incredibly difficult to keep them from getting administrative access; operating systems and the privileged applications that are generally found on them constitute such a large and complex body of code that the presence of some security hole unknown to the masses (or at least the system administrator) is always likely.

Nonetheless, both kinds of problems are important, and it is important for companies wishing to be secure to keep up with security vulnerabilities in software. There are several popular sources for vulnerability information:

Bugtraq. The Bugtraq mailing list, administered by securityfocus.com, is an e-mail discussion list devoted to security issues. Many security vulnerabilities are proposed, scripted, and patched on Bugtraq (which generates a large amount of traffic). The signal to noise ratio on Bugtraq is low, so reader discretion is advised. Nevertheless, this list is often the source of breaking security news and interesting technical discussion. Plenty of computer security reporters use Bugtraq as their primary source of information.

Bugtraq is famous for pioneering the principle of "full disclosure", which is a debated notion that making full information about security vulnerabilities public will encourage vendors to fix such problems more quickly. This philosophy was driven by numerous documented cases of vendors downplaying security problems, refusing to fix them when they believed that few of their customers would find out about any problems.

If the signal to noise ratio on Bugtraq is too low for your tastes, the securityfocus.com web site keeps good information on recent vulnerabilities.

CERT Advisories. The CERT Coordination Center (CERT/CC, www.cert.org) is located at the Software Engineering Institute, a federally funded research and development center operated by Carnegie Mellon University. CERT/CC studies Internet security vulnerabilities, provides incident response services to sites that have been the victims of attack, and publishes a variety of security alerts.

Many people in the security community complain that CERT/CC announces problems much too late to be effective. For example, a problem in the TCP protocol was the subject of a CERT advisory released a decade after the flaw was first made public. Delays experienced in the past can largely be attributed to the (since changed) policy of not publicizing an attack until patches were available. However, problems like the aforementioned TCP vulnerability are more likely attributed to the small size of CERT. CERT tends only to release advisories for significant problems. In this case, they reacted once the problem was being commonly exploited in the wild. The advantage of CERT/CC as a knowledge source is that they highlight attacks that are in actual use, and

ignore low-level malicious activity. That makes CERT a good source for making risk management decisions and working on problems that really matter.

RISKS Digest. The RISKS Digest forum is a mailing list compiled by security guru Peter Neumann that covers all kinds of security, safety, and reliability risks introduced and exacerbated by technology. RISKS Digest is often among the first places that sophisticated attacks discovered by the security research community are announced. Most Java Security attacks, for example, first appeared here. The preferred method for reading the RISKS Digest is through the Usenet News group comp.risks. However, for those without easy access to Usenet News, you can subscribe to the mailing list by sending a request to risks-request@CSL.SRI.COM.

These aren't the only sources for novel information, but they're certainly among the most popular. The biggest problem isthat there are too many sources. To get the "big picture", one must sift through too much information. Often, administrators never learn of the existence of an important patch that should definitely be applied to their system. We don't really consider this problem the fault of the system administrator. As Bruce Schneier has said, blaming a system administrator for not keeping up with patches is blaming the victim. It is very much like saying, "you deserved to get mugged for being out alone in a bad neighborhood after midnight." Having to keep up with dozens of weekly reports announcing security vulnerabilities is a Herculean task that is also thankless. People don't see the payoff.

Technical Trends Affecting Software Security

Complex systems, by their very nature, introduce multiple risks. And almost all systems that involve software are complex. One risk is that malicious functionality can be added to a system (either during creation or afterwards) that extends it past its primary intended design. As an unfortunate side effect, inherent complexity lets malicious and flawed subsystems remain invisible to unsuspecting users until it is too late. This is one of the root causes of the malicious code problem [McGraw, 2000]. Another risk, more relevant to our purposes, is that the complexity of a system makes it hard to understand, hard to analyze, and hard to secure. Security is difficult to get right even in simple systems; complex systems serve only to make security harder. Security risks can remain hidden in the jungle of complexity, not coming to light until it is too late.

Extensible systems, including computers, are particularly susceptible to complexity-driven hidden risk and malicious functionality problems. When extending a system is as easy as writing and installing a program, the risk of intentional introduction of malicious behavior increases drastically; so does the risk of introducing unintentional vulnerabilities.

Any computing system is susceptible to hidden risk. Rogue programmers can modify systems software that is initially installed on the machine. Unwitting programmers may introduce a security vulnerability when adding important features to a network-based application. Users may incorrectly install a program that introduces unacceptable risk, or worse yet accidentally propagate a virus by installing new programs or software updates.

In a multi-user system, a hostile user might install a Trojan Horse to collect other users' passwords. These attack classes have been well known since the dawn of computing, so why is software security a bigger problem now than in the past? We believe that a small number of trends have a large amount of influence on the software security problem.

One significant problem is the fact that computer networks are becoming ubiquitous. The growing connectivity of computers through the Internet has increased both the number of attack vectors (avenues for attack) and the ease with which an attack can be made. More and more computers, ranging from home PCs to systems that control critical infrastructures (such as the power grid), are being connected to the Internet. Furthermore, people, businesses, and governments are increasingly dependent upon network-enabled communication such as e-mail or Web pages provided by information systems. Unfortunately, as these systems are connected to the Internet, they become vulnerable to attacks from distant sources. Put simply, an attacker no longer needs physical access to a system to cause security problems.

Because access through a network does not require human intervention, launching automated attacks from the comfort of your living room is relatively easy. Indeed, the well-publicized denial-of-service attacks in February of 2000 took advantage of a number of (previously compromised) hosts to flood popular e-commerce Web sites, including Yahoo!, with bogus requests automatically. The ubiquity of networking means that there are more systems to attack, more attacks, and greater risks from poor software security practice than ever before.

A second trend that has allowed software security vulnerabilities to flourish is the size and complexity of modern information systems and their corresponding programs. A desktop system running Windows/NT and associated applications depends upon the proper functioning of the kernel as well as the applications to ensure that an attacker cannot corrupt the system. However, NT itself consists of approximately 35 million lines of code, and applications are becoming equally, if not more, complex. When systems become this large, bugs cannot be avoided.

Exacerbating this problem is the widespread use of low-level programming languages such as C or C++ that do not protect against simple kinds of attacks (most notably, buffer overflows). However, even if the systems and applications code were bug free, improper configuration by retailers, administrators, or users can open the door to attackers. In addition to providing more avenues for attack, complex systems make it easier to hide or mask malicious code. In theory, we could analyze and prove that a small program was free of security problems, but this task is impossible for even the simplest desktop systems today, much less the enterprise-wide systems used by businesses or governments.

A third trend exacerbating software security problems is the degree to which systems have become extensible. An extensible host accepts updates or extensions, sometimes referred to as mobile code, so that the system's functionality can be evolved in an

incremental fashion. For example, the plug-in architecture of Web browsers makes it easy to install viewer extensions for new document types as needed.

Given a basic intuitive grasp of the architecture of a browser (a program that runs on top of an operating system and provides basic Web interface services), it is natural to assume that a browser might be used to enhance security. In reality, it is hard to tell where the boundaries of the browser are, where the operating system fits in, and how a browser can protect itself. The two most popular browsers, Netscape Navigator and Microsoft Internet Explorer, have very fuzzy boundaries and include many more hooks to external applications than most people realize.

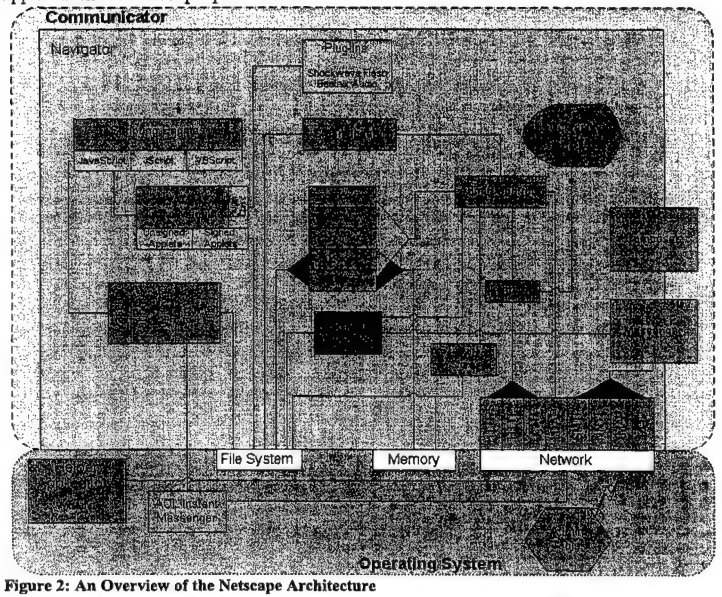


Figure 2: An Overview of the Netscape Architecture

On the most pervasive platform (Windows 95, 98, ME) there is really no way for a browser to protect itself or any secrets it may be trying to keep (like client side certificates) at all. That means if your design requires some security features inside the browser (like an intact Java Virtual Machine or a cryptographic secret) there is probably a real need for a more advanced operating system like Windows/NT or Unix.

Without delving into the details of how a browser is constructed, it is worth showing a general purpose abstract architectural diagram of each. Figures 2 and 3 show Netscape and MSIE, respectively. From a high level perspective, it is clear that there are many

interacting components involved in each architecture. That makes securing a browser quite a monumental task. In addition, helper applications (such as AOL Instant Messenger for Netscape and ActiveX control functionality in MSIE) introduce large security risks.

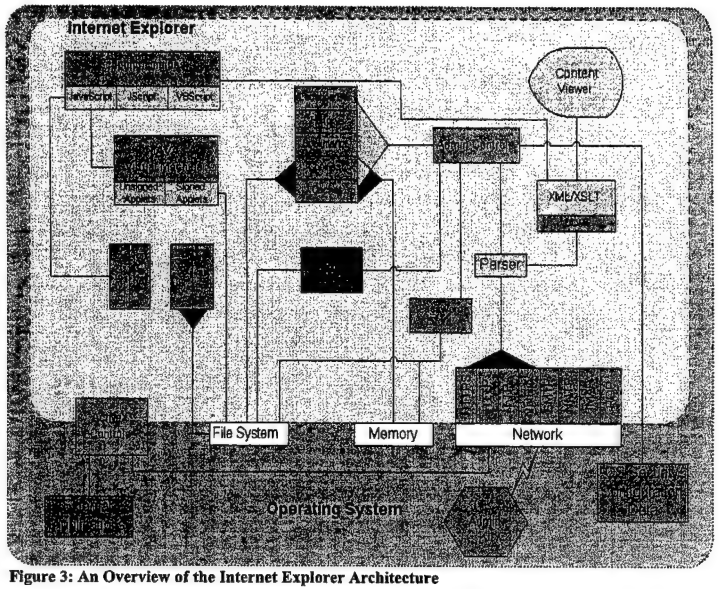


Figure 3: An Overview of the Internet Explorer Architecture

Browsers are not the only extensible systems. Today's operating systems support extensibility through dynamically loadable device drivers and modules. Today's applications, such as word processors, e-mail clients, spreadsheets, and (yes) Web browsers, support extensibility through scripting, controls, components, dynamically loadable libraries, and applets.

From an economic standpoint, extensible systems are attractive because they provide flexible interfaces that can be adapted through new components. In today's marketplace, it is crucial that software be deployed as rapidly as possible to gain market share. Yet the marketplace also demands that applications provide new features with each release. An extensible architecture makes it easy to satisfy both demands by letting companies ship the base application code early and later ship feature extensions as needed.

Unfortunately, the very nature of extensible systems makes security harder. For one thing, it is hard to prevent malicious code from slipping in as an unwanted extension,

meaning the features designed to add extensibility to a system (such as Java's class loading mechanism) must be designed with security in mind. Furthermore, analyzing the security of an extensible system is much harder than analyzing a complete system that can't be changed. How can you take a look at code that has yet to arrive? Better yet, how can you even begin to anticipate every kind of mobile code that may arrive?

Together, the three trends of ubiquitous networking, growing system complexity, and built-in extensibility make the software security problem more urgent than ever. There are other trends that have an impact as well, such as the lack of diversity in popular computing environments, and the tendency for people to use systems in unintended ways (e.g., using Windows NT as an embedded operating system). For more on security trends, see Bruce Schneier's book, *Secrets and Lies* [Schneier, 2000].

The 'ilities

Is security a feature that can be added on to an existing system? Is it a static property of software that remains the same no matter what environment the code is placed in? The answer to these questions is an emphatic *no*.

Bolting security onto an existing system is simply a bad idea. Security is not a feature you can add to a system at any time. Security is like safety, dependability, reliability, or any other software *ility*. Each ility is a system-wide emergent property that requires advance planning and careful design. Security is a behavioral property of a complete system in a particular environment.

It is always better to design for security from scratch than to try to add security to an existing design. Reuse is an admirable goal, but the environment in which a system will be used is so integral to security that any change of environment is likely to cause all sorts of trouble—so much trouble that well-tested and well-understood software can simply fall to pieces.

We have come across many real-world systems (designed for use over protected proprietary networks) that were being reworked for use over the Internet. In every one of these cases, Internet-specific risks caused the systems to lose all their security properties. Some people refer to this problem as an *environment problem*, where a system that is secure enough in one environment is completely insecure when placed in another. As the world becomes more interconnected via the Internet, the environment most machines find themselves in is at times less than friendly. A good example of this problem is the way links to financial institutions have traditionally been secured. Such connections typically have security that is quite poor compared to what we've come to expect from the Internet, primarily because the original designers of those systems never considered that they might be put on a large network. Nonetheless, we have seen plenty of people try, including people who do not fully understand the security implications of what they are doing.

What Is Security?

So far, we have dodged the question we often hear asked, "What is security?" Security means different things to different people. It might even mean different things to the same person, depending on the context. For us, security boils down to enforcing a policy that describes rules for accessing resources. If we don't want unauthorized users logging into our system, and they do, then we have a security violation on our hands. Similarly, if someone performs a denial of service attack against us, then they're probably violating our policy on acceptable availability of our server or product. In many cases, we don't really require an explicit security policy since our implicit policy is fairly obvious and widely shared.

Without a well-defined policy, however, arguing whether some event is really a security breach can become difficult. Is a port scan considered a security breach? Do you need to take steps to counter such "attacks"? There's no universal answer to this question. Despite wide evidence of such gray areas, most people tend to have an implicit policy that gets them pretty far. Instead of disagreeing on whether a particular action someone takes is a security problem or not, we worry about things like whether the consequences are significant or whether there is anything we can do about the potential problem at all.

Isn't that just reliability?

Comparing reliability with security is a natural thing to do. At the very least, reliability and security have a lot in common. Reliability is roughly a measurement of how robust your software is with respect to some definition of a bug. The definition of a bug is in analogous to a security policy. Security can be seen as a measurement of how robust your software is with respect to a particular security policy. Some people argue that security is a subset of reliability, and some argue the reverse. We're of the opinion that security is a subset of reliability. If you manage to violate a security policy, then there's a bug. The security policy always seems to be part of the particular definition of "robust" that is applied to a particular product.

Reliability problems aren't always security problems, though we should note that reliability problems are security problems a lot more often than one might think. For example, sometimes bugs that can crash a program provide a potential attacker with unauthorized access to a resource. However, reliability problems can usually be considered denial of service problems. If an attacker knows a good, remotely exploitable "crasher" in the web server you're using, this can be leveraged into a denial of service attack by tickling the problem as often as possible.

If you apply solid software reliability techniques to your software, you will probably improve its security, especially against some kinds of an attack. Therefore, we recommend to anyone wishing to improve the security of their products to also work on improving the overall robustness of their products. We won't cover that kind of material in any depth in this book; there are several good books on software reliability and testing, including the two classics, *Software Testing Techniques* by Boris Beizer [Beizer, 1990] and *Testing Computer Software* by Cem Kaner et. al. [Kaner, 1999]. We also recommend picking up a good text on software engineering, such as [Hamlet, 2001].

Penetrate and Patch is Bad

Many well-known software vendors don't yet understand that security is not an add-on feature. They continue to design and create products at alarming rates, with little attention paid to security. They start to worry about security only after their product has been publicly (and often spectacularly) broken by someone. And then they rush out a patch instead of coming to the realization that designing security in from the start might be a better idea. This sort of approach won't do in e-commerce or other business-critical applications.

Our goal is to minimize the unfortunately pervasive "penetrate and patch" approach to security, and avoid the problem of desperately trying to come up with a fix to a problem that is being actively exploited by attackers. In simple economic terms, finding and removing bugs in a software system before its release is orders of magnitude cheaper and more effective that trying to fix systems after release [Brooks, 1995].

There are many problems to the penetrate-and-patch approach to security, among them:

- Developers can only patch problems that they know about. Attackers may find problems that they never report to developers.
- Patches are rushed out as a result of market pressures on vendors, and often introduce new problems of their own to a system.
- Patches often only fix the symptom of a problem, and do nothing to address the underlying cause.
- Patches often go unapplied, as system administrators tend to be overworked, and often do not wish to make changes to a system that "works". As we discussed above, system administrators are generally not security professionals.

Designing a system for security, carefully implementing the system, and testing the system extensively before release, presents a much better alternative. We will discuss design for security extensively in Chapter 2.

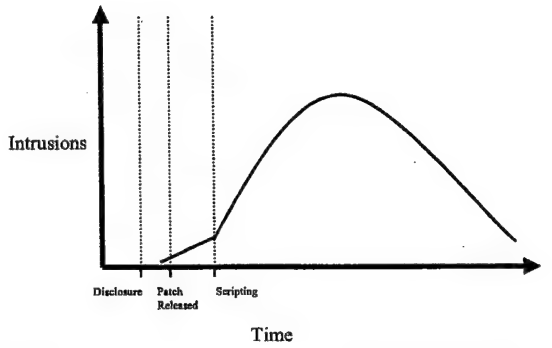


Figure 4: Average curve of number of intrusions for a security bug over time.

The fact that the existing penetrate and patch system is so poorly implemented is yet another reason why the approach needs to be changed. In an IEEE Computer article from December 2000 entitled "Windows of Vulnerability: A Case Study Analysis", Bill Arbaugh, Bill Fithen, and John McHugh discuss a life-cycle model for system vulnerabilities that emphasizes how big the problem is [Arbaugh, 2000]. Data from their study show that intrusions increase once a vulnerability is discovered, the rate continues to increase until the vendor releases a patch, but exploits continue to occur even after the patch is issued (sometimes years after). Figure 4 is based on their data. It takes a long time before most people upgrade to patched versions, because most people upgrade for newer functionality, the hope of more robust software or better performance, not because they know of a real vulnerability.

On Art and Engineering

Software that is properly engineered goes through a well-structured process from requirements design, through detailed specification, to actual implementation. In the world of consumer-ware (software created for the mass consumer market, like browsers), pressure to be first to market and retain what is known as "mind share" compresses the development process so much that software engineering methods are often thrown out the window. This is especially true of testing, which regularly ends up with no scheduled time and few resources. An all too common approach is to leave rigorous testing to users in the field (sometimes even paying users when they find bugs!). We think this is just awful.

The *Internet time* phenomenon has exacerbated the software engineering problem. These days, Internet years rival dog years in shortness of duration. So three months of regular

time are currently equivalent to a complete Internet "year." Given the compressed development schedules that go along with this accelerated kind of calendar, the fact that specifications are often very poorly written (if they exist at all) is not surprising. It is not uncommon to encounter popular consumer-oriented systems that have no specifications.

Java makes a good case study of the complex interrelation between secure design and secure implementation. One of the most common misconceptions about Java security holes is that they are all simple implementation errors and that the specification has been sound and complete since day one. Threads in the newsgroup comp.lang.java.security and other newsgroups often repeat this fallacy as people attempt to trivialize Java's security holes. The truth is that many of the holes described in books like *Securing Java* are simple implementation bugs (the code-signing hole from April 1997 comes to mind), but others, like problems discovered in Java class loaders, are not [McGraw, 1999]. Sometimes the specification is just plain wrong and must be changed. As an example, consider the Java specification for class loading, which has evolved as inadequacies have come to light. In any case, the much hyped Java security model focuses primarily on protecting against malicious mobile code. Java is still susceptible to a large number of the same problems other languages are.

Often it is hard to determine whether a security hole is an implementation problem or a specification problem. Specifications are notoriously vague. Given a vague specification, who is to blame when a poor implementation decision is made? Specifications are also often silent; that is, when a hole is discovered and the specification is consulted, there is nothing said about the specific problem area. These sorts of omissions certainly lead to security problems, but are the resulting problems specification problems or implementation problems? In the end, the holes are fixed, regardless of whether they are implementation bugs or design-level problems. This leads to a more robust system. Of course, designing, implementing, and testing things properly in the first place is the least expensive and most successful approach. We discuss how to use software engineering to do these things in Chapter 2.

Security Goals

What does it mean for a software system to be secure? Does it even make sense to make claims like "Java is secure"? How can a program be secure?

Security is not a static feature that everyone agrees on. It's not something than can be conveniently defined away in a reductionist move. For most developers and architects, security is like pornography is to the United States Supreme Court: they may not be able to define it, but they think they know it when they see it.

The problem is, security is relative. Not only is there no such thing as 100% security, even figuring out what "secure" means differs according to context. A key insight about security is to realize that any given system, no matter how "secure" can probably be broken. In the end, security must be understood in terms of a simple question: *Secure against what and from who?*

Understanding security is best understood by thinking about goals. What is it we are trying to protect? Who are we protecting it from? How can we get what we want?

Prevention

As in today's criminal justice system, much more attention is paid to security after something bad happens than before. In both cases, an ounce of prevention is probably worth a pound of punishment.

Internet time compresses not only the software development lifecycle (making software risk management a real challenge), it also directly affects the propagation of attacks. Once a successful attack on a vulnerability is found, the attack spreads like wildfire on the Internet. Often, the attack is embedded in a simple script, so that an attacker requires no more skill than the ability to hit return in order to carry it out.

Internet time is the enemy of software security. Automated Internet-based attacks on software are a serious threat that must be factored into the risk management equation. This makes prevention more important than ever.

Traceability and Auditing

Since there is no such thing as 100% security, attacks will happen. One of the keys to recovering from an attack is to know who did what, and when they did it. Though auditing is not a direct prevention technology, knowing that there is a system for accountability may in some cases dissuade potential attackers.

Auditing is well understood by accountants, who have practiced double entry bookkeeping for over 500 years. Banks and other financial institutions have entire divisions devoted to auditing. Most businesses audit their inventories. Every public company has its books audited by a designated accounting firm in order to meet SEC regulations. Any system in which security is important should seriously consider including auditing.

Good auditing and traceability measures are essential for forensics. They help detect, dissect, and demonstrate an attack. They show who did what when, and provide critical evidence in court proceedings.

Software auditing is a technological challenge. Bits stored on disk as an audit log are themselves susceptible to attack. Verification of an audit log is thus tricky. Nevertheless, auditing is an essential part of software security.

Monitoring

Monitoring is real-time auditing. Intrusion detection (ID) systems based on watching network traffic or poring over logfiles are simple kinds of monitoring systems. ID systems are relative newcomers to the commercial security world, and getting the shrinkwrap products to be useful is not easy due to the alarming number of false alarms.

Monitoring a program is possible on many levels, and is an idea rarely practiced today. Simple approaches can watch for known signatures, such as dangerous patterns of low level system calls that identify an attack in progress. More complex approaches place monitors in the code itself in the form of assertions.

Often, simple burglar alarms and tripwires can catch an attack in progress and help prevent serious damage.

Privacy and Confidentiality

Privacy and confidentiality are deeply intertwined. There are clear reasons for business, individuals, and governments to keep secrets. Businesses must protect trade secrets from competitors. Web users often want to protect their on-line activities from the invasive marketing machines of AOL, Amazon.com, Yahoo!, and DoubleClick. Governments have classified military secrets to protect.

Of these three groups, individuals probably least understand how important privacy can be. But they are beginning to clue in. Privacy groups such as the Privacy Foundation (www.privacyfoundation.org), and the Electronic Privacy Information Center (www.epic.org), are beginning to improve the awareness of the general public. Interestingly, the European Union has a large head start over the US in terms of privacy laws.

In any case, there are often lots of reasons for software to keep secrets and ensure privacy. The problem is, software is not really designed to do this. Software is designed to run on a machine and accomplish some useful work. That means the machine a program is running on can pry out every secret a piece of software may be trying to hide.

One very simple, useful piece of advice is to avoid storing secrets like passwords in your code, especially if that code is likely to be mobile.

Multi-level security

Some kinds of information are more secret than others. Most governments have multiple levels of information classification, ranging from Unclassified and merely For Official Use Only, through Secret and Top Secret, all the way to Top Secret/Special Compartmentalized Intelligence.

Most corporations have data to protect too, sometimes from their own employees. Having a list of salaries floating around rarely makes for a happy company. Some business partners will be trusted more than others. Technologies to support these kinds of differences are not as mature as we might wish.

Different levels of protection are afforded different levels of information. Getting software to interact cleanly with a multi-level security system is tricky.

Anonymity

Anonymity is a double-edged sword. Often there are good social reasons for some kinds of anonymous speech (think of AIDS patients discussing their malady on the Internet), but just as often there are good social reasons not to allow anonymity (think of hate speech, terrorist threats, and so on). Today's software often makes inherent and unanticipated decisions about anonymity. Together with privacy, decisions about anonymity are important aspects of software security.

Microsoft's Global Identifier tracks which particular copy of Microsoft Office originated a document. Sound like Big Brother? Consider that this identifier was used to help tie David L. Smith, the system administrator who created and released the Melissa virus, to his malicious code.[2] What hurts us can help us too.

Often, technology that severely degrades anonymity and privacy turns out to be useful for law enforcement. The FBI's notorious Carnivore system is set up to track who sends e-mail to whom by placing a traffic monitoring system at an ISP (like AOL). But why should we trust the FBI to stop at the headers? Especially when we know that the supposedly secret Echelon system (also run by the US government) regularly scans international communications for keywords and patterns of activity!

Cookies are used with regularity by e-commerce sites wanting to learn more about the habits of their customers. Cookies make buying airline tickets on-line faster and easier, and they remember your Amazon.com identity so you don't have to type in your username and password every time you want to buy a book. But cookies can cross the line when a single collection point is set up to link cross-Web surfing patterns. DoubleClick collects reams of surfing data about individuals across hundreds of popular Websites, and they have publicly announced their intention to link surfing data in with other demographic data. This is a direct marketer's dream, and a privacy advocate's nightmare.

Software architects and developers, along with their managers, should think carefully about what might happen to data they collect in their programs. Can the data be misused? How? Does convenience outweigh potential privacy issues?

Authentication

Authentication is held up as one of the big three security goals (the other two being confidentiality and integrity). Authentication is crucial to security, because it is essential to know who to trust and who not to trust. Enforcing a security policy of almost any sort requires knowing *who* it is that is trying to do something to the *what* we want to protect.

Software security almost always includes authentication issues. Most security-critical systems require a user to log in with a password before they can do anything. Then, based on the user's role in the system, he or she may be disallowed from doing certain things.

[2] Steve Bellovin tells us that, while the GUID was found, David Smith was first tracked down via phone records and ISP logs.

Not too many years ago, PCs did very little in the way of authentication. Physical presence was good enough for the machine to let you do anything. This simplistic approach fails to work in a networked world.

Authentication on the Web is in a sorry state these days. Users tend to trust that a URL displayed on the status line means they are looking at a Web site owned by a particular business or person. But a URL is no way to spread around trust! Who's to say that yourfriendlybank.com is really a bank, and is really friendly? Does united.com belong to the airline we all love to hate?

People falsely believe that when the little lock icon on their browser lights up that they have a "secure connection". SSL uses cryptography to protect the datastream between the browser and the server it is connected to. But from an authentication standpoint, the real question to ponder is *who* are you connected to? Clicking on the little key may reveal a surprise.

When you buy an airline ticket from ual.com (the real United Airlines site as far as we can tell), a secure SSL connection is used. Presumably this secures your credit card information on its travels across the Net. But click on the lock and you'll see that the site you have a secure channel with is not ual.com, it's itn.net (and the certificate belongs to GetThere.com of Menlo Park, CA). Who the heck are they? Can they be trusted with your credit card data?

To abuse SSL in a Web spoofing attack (as described in [Felten, 1997]), a bad guy must establish a secure connection with the victim's browser at the right times. Most users never bother to click the lock (and sophisticated attackers can make the resulting screens appear to say whatever they want anyway).

Authentication in software is a critical software security problem to take seriously. And there will be literally hundreds of different ways to solve it (see Chapter 3). Stored-value systems and other financial transaction systems will require very strong approaches. Loyalty programs like frequent flyer programs won't. Some authentication schemes will require anonymity, and others strict and detailed auditing. Some schemes will involve sessions (logging in to your computer in the morning) while others will be geared toward transactions (payment systems).

Integrity

Last but certainly not least comes integrity. When used in a security context, integrity refers to staying the same. By contrast to authentication, which is all about who, when, and how, integrity is about whether something has been modified since its creation.

There are many kinds of data that people rely on to be correct. Stock prices are a great example. The move towards Internet-enabled devices like WAP phones or iMode devices is often advertised with reference to real time trading decisions made by a busy person watching the stock market on their phone as they walk through the airport or hear a stock

holder's presentation. What if the data are tampered with between the stock exchange (where the information originates) and the receiver?

Stock price manipulation via misinformation is more common than you might think. Famous cases include Pairgain Technologies, whose stock was intentionally run up in 1999 by a dishonest employee, and Emulex Corporation, a fiber channel company whose stock price was similarly manipulated with fake wire stories by a junior college student in California.

Digital information is particularly easy to fake. Sometimes it can be harder to print counterfeit money than to hack a stored-value smart card with differential power analysis (DPA) and add some electronic blips (see the book web site for links on DPA). The more the new economy comes to rely on information, the more critical information integrity will become.

Know Your Enemy: Common software security pitfalls

There are many excellent software developers and architects in the world building all sorts of exciting things. The problem is that though these developers and architects are world class, they have not had the opportunity to learn much about security. Part of the issue is that most security courses at universities emphasize network security (that is, if the university teaches anything about security at all). Very little attention is paid to building secure software. Another part of the problem is that though some information exists covering software security, a comprehensive, practical guide to the topic has never really been available. We hope to change all that with this book.

The aphorism, "keep your friends close and your enemies closer," applies quite aptly to software security. Software security is really risk management. The key to an effective risk assessment is expert knowledge of security. Being able to recognize situations where common attacks can be applied is half the battle. The first step in any analysis is recognizing the risks.

Software security risks come in two main flavors: architectural problems and implementation errors. We'll cover both kinds of security problems and their associated exploits throughout the book. Most software security material focuses on the implementation errors leading to problems such as buffer overflows (Chapter 7), race conditions (Chapter 9), randomness problems (Chapter 10), and a handful of other common mistakes. These issues are important, and we'll devote plenty of space to them

But there is more to software security than avoiding the all-too-pervasive buffer overflow. Building secure software is like building a house. The kinds of bricks you use are important, but even more important (if you want to keep bad things out) is having four walls in the design. The same thing goes for software: what system calls you use and how you use them is important, but overall design properties often count for more. We devote Part I of this book to issues that primarily apply at design time, including integrating security into your software engineering methodology, general principles for

developing secure software systems, and dealing with security when performing security assessments.

Threats Against Software

It is important to understand what kinds of threats your software will face. At a high level, the answer is more or less any threat you can imagine from the real world. Theft, fraud, break-ins, and vandalism are all alive and well on the net.

Getting a bit more specific, we should mention a few of the more important types of threats you should be wary of. One significant category of high-level threat is the compromise of information as it passes through or resides on each node in a network. Client-server models are commonly encountered in today's software systems, but things can get arbitrarily more complex. The kinds of attacks that are possible at each node are virtually limitless. We will spend much of the book discussing different kinds of attacks. The attacks that can be launched vary, depending on the degree to which we trust the people at each node on the network.

One important kind of problem that isn't necessarily a software problem per-se is *"social engineering"*, which involves talking someone into giving up important information leading to the compromise of a system through pure charisma and chutzpah. Attackers often get passwords changed on arbitrary accounts just by calling up technical support, sounding angry, and knowing some easily obtained (usually public) information. See Ira Winkler's book *Corporate Espionage* for more information on social engineering [Winkler, 1997].

On the server side of software security, many threats boil down to *malicious input problems*. Chapter 12 is devoted to this problem. *Buffer overflows* (discussed in Chapter 7) are probably the most famous sort of malicious input problem.

Besides worrying about the nodes in a data communication topology, we have to worry about data being compromised on the actual communication medium itself. While some people assume that such attacks aren't possible (perhaps because they don't know how to perform them), network-based attacks actually turn out to be relatively easy in practice. Some of the most notable and easiest to perform network attacks include:

- Eavesdropping. The attacker watches data as they traverse a network. Such attacks are sometimes possible, even when using strong cryptography. See the discussion on "man-in-the-middle" attacks in Appendix A.
- Tampering. The attacker maliciously modifies data that are in-transit on a network.
- Spoofing. The attacker generates phony network data to give the illusion that valid data are arriving, when in reality the data are bogus.
- Hijacking. The attacker replaces a stream of data on a network with his or her own stream of data. For example, once someone has authenticated a remote connection using telnet, an attacker can take over the connection by killing

packets from the client, and submitting "attack" packets. Such attacks generally involve spoofing.
- Capture-replay. An attacker records a stream of data, and later sends the exact same traffic in an attempt to repeat the effects, with undesirable consequences. For example, an attacker might capture a transaction in which someone sells 100 shares of Microsoft stock. If the victim had thousands more, an attacker could wait for the stock price to dip, then replay the sale *ad nasuem* until the target had none remaining.

Cryptography can be used to solve these network problems to a varying degree. For those without exposure to cryptography basics, we provide an overview in Appendix A. In Chapter 11, we look at the practical side of using cryptography in applications.

Our simple list of threats is by no means comprehensive. What we present are some of the most important and salient concepts that we want to expose you to up front. We will discuss each of these threats, and many more, in detail throughout the book.

Software Project Goals

The problem with the security goals we discussed above is that they directly clash with many software project goals. What are these goals like, and why are they important? And what on earth could be more important than security (ha ha)?

We won't spend much time discussing software project goals in great detail here, as they are covered in many other places. But mentioning some of the more relevant is worth a few words.

Key software project goals include:

- **Functionality** This is the number one driver in most software projects, which are obsessed with meeting functional requirements. And for good reason too. Software is a tool meant to solve a problem. Solving the problem invariably involves getting something done in a particular way.

- **Usability** People like programs to be easy to use, especially when it comes to conducting electronic commerce. Usability is very important, because without it software becomes too much of a hassle to actually work with. Usability affects reliability too, since human error often leads to software failure.

  The problem is, security mechanisms, including most uses of cryptography, elaborate login procedures, tedious audit requirements, and so on, often cause usability to plummet. Security concerns regularly impact convenience too. Security people often deride cookies, but cookies are a great user-friendly programming tool!

- **Efficiency** People tend to want to squeeze every ounce out of their software (even when efficiency isn't needed). Efficiency can be an important goal, though it usually trades off against simplicity. Security often comes with significant

overhead. Waiting around while a remote server somewhere authenticates you is no fun. But it can be necessary.

- **Time to market** "Internet time" happens. Sometimes the very survival of an enterprise requires getting mind share fast in a quickly evolving area.

  Unfortunately, the first thing to go in a software project with severe time constraints is any attention to software risk management. Design, analysis, and testing corners are cut with regularity. This often introduces grave security risks.

  Fortunately, building secure software does not have to be slow. Given the right level of expertise, a secure system can sometimes be designed more quickly than an *ad hoc* cousin system with little or no security.

- **Simplicity** The good thing about simplicity is that it is a good idea for both software project *and* security. Everyone agrees that keeping it simple is good advice.

Software Security means Good Software

Computer security is a vast topic that is becoming more important because the world is becoming highly interconnected, with networks being used to carry out critical transactions. The environment in which machines must survive has changed radically since the popularization of the Internet. Deciding to connect a LAN to the Internet is a security-critical decision. The root of most security problems is software that fails in unexpected ways. Though software security as a field has much maturing to do, it has much to offer to those practitioners interested in striking at the heart of security problems. The goal of this book is to verse you in the current best practices for keeping security flaws out of your software.

Good software security practices can help ensure that software behaves properly. Safety-critical and high assurance system designers have always taken great pains to analyze and track software behavior. Security-critical system designers must follow suit. We can avoid the band-aid–like penetrate-and-patch approach to security only by considering security as a crucial system property. This requires integrating software security into your entire software engineering process, a topic that we take up in the next chapter.

SIDEBAR

Hackers, Crackers, and Attackers

We'll admit it; we are hackers. But don't break out the handcuffs yet; we don't run around breaking into machines, reading other people's e-mail, and erasing hard disks. In fact, we stay firmly on this side of the law (well, we would in a world where driving really fast is always legal).

The term "hacker" originally had a positive meaning. It sprang from the computer science culture at MIT in the late 1960's, where it was used as a badge of honor to refer to people who were exceptionally good at solving tricky problems through programming, often as if by magic. For most people in the Unix development community, the term's preferred definition retains that meaning, or is just used to refer to someone who is an excellent and enthusiastic programmer. Often, hackers like to tinker with things, and figure out how they work.

Software engineers commonly use the term "hacker" in a way that carries a slightly negative connotation. To them, a "hacker" is still the programming world's equivalent of MacGyver; someone who can solve hard programming problems if given only a ball of fishing twine, a matchbook and two sticks of chewing gum. The problem for software engineers is not that "hackers" (by their definition) are malicious; it is that they believe cobbling together an *ad hoc* solution is at best barely acceptable. They feel careful thought should be given to software design before coding begins. They therefore feel that hackers are developers who tend to "wing it", instead of using sound engineering principles. (Of course, *we* never do that! Wait, did we say that we're hackers?)

Far more negative is the definition of "hacker" normal people use (including the press). To most people, a hacker is someone who maliciously tries to break software. If someone breaks onto your machine, many people would call that person a "hacker". Needless to say, this definition is one that the many programmers who consider themselves "hackers" resent.

Do we call locksmiths burglars just because they could break into our house if they wanted to do so? Of course not. But that's not to say that no locksmiths can be burglars. And, of course, there are hackers who are malicious, and do break into other people's machines, and do erase disk drives. These people are a very small minority compared to the number of expert programmers who consider themselves "hackers".

In the mid 80's, people who considered themselves "hackers", but hated the negative connotations the term carried in most peoples minds (mainly due to media coverage of security problems), coined the term "cracker". A "cracker" is someone who breaks software for nefarious ends.

Unfortunately, this term hasn't caught on much outside of hacker circles. The media doesn't use it, probably because it is already quite comfortable with "hacker". And it sure didn't help that they called the movie "Hackers" instead of "Crackers". Nonetheless, we think it is insulting to lump all hackers together under a negative light. But we don't like the term "cracker" either. To us, it sounds dorky, bringing images of saltines to mind. So when we use the term "hacker", that should give you warm fuzzy feelings. When we use the term "malicious hacker", "attacker" or "bad guy", it is okay to scowl. If we say "malicious hacker", we're generally implying that the person at hand is skilled. If we say anything else, they may or may not be.

Who is the bad guy?

We've said that some hackers are malicious. Many of the ones who break software protection mechanisms so that pirate copies can be easily distributed are. Removing such protection mechanisms takes a fair bit of skill, which is probably just cause to label that particular person a hacker as well. However, most bad guys are not hackers, they're just kids trying to break into other people's machines.

Some hackers who are interested in security may take a piece of software and try to break it just out of sheer curiosity. If they do break it, they won't do anything malicious; they will instead notify the author of the software, so that the problem can be fixed. If they don't tell the author in a reasonable way, then they can safely be labeled malicious. Fortunately, most people that find serious security flaws don't use their finds to do bad things.

At this point, you are probably wondering who the malicious attackers and bad guys really are! After all it takes someone with serious programming experience to break software. That may be true in finding a new exploit, but not in exploiting it. Generally, bad guys are people with little or no programming ability capable of downloading, building and running programs other people write (hackers often call this sort of bad guy a "script kiddie"). These kinds of bad guys go to a hacker site, such as (the largely defunct) http://www.rootshell.com, download a program that can be used to break onto a machine, and get the program to run. It doesn't take all that much skill (other than hitting return a few times). Most of the people we see engage in this sort of activity tend to be teenage males going through a rebellious period. Despite the fact that such people tend to be relatively unskilled, they are still very dangerous.

So you might ask yourself, who wrote the programs these script kiddies use to break stuff? Hackers? And if so, then isn't that highly unethical? Yes, hackers tend to write most such programs. Some of those hackers do indeed have malicious intent, and are truly bad people. However, many of these programs are made public by hackers who believe in the principle of "full disclosure". The basic idea behind this principle is that everyone will be encouraged to write more secure software if all security vulnerabilities in software are made public.

In the eyes of full disclosure zealots, developers that are able to look at other people's security problems can hopefully avoid them in their own code. Often, trying to explain the gist of a security problem isn't enough to help a developer. Without a working example of what an attacker might do, the developer won't quite understand the problem. That's one good reason for full disclosure. Full disclosure also allows informed administrators to address problems in software before the software vendor gets around to releasing a patch. Usually, the full disclosure people will actually release slightly broken exploits, in the hopes of keeping the untalented masses from using it for evil. However, it only takes a script kiddie with an iota of true talent to undo that effort.

Of course, there are plenty of arguments against full disclosure too. While full disclosure may encourage vendors to fix their problems quickly, there is almost always a long delay

before all users of a vendor's software upgrade. Giving so much information to potential attackers makes it much easier for them to exploit those people. In the year 2000, Marcus Ranum sparked considerable debate on this issue, arguing that full disclosure does more harm than good. Suffice it to say, people without malicious intent sometimes do release working attack code on a regular basis, knowing that there is a potential for their code to be misused. These people believe the benefits of full disclosure outweigh the risks. We provide third-party information on these debates on the book's web site.

Building Secure Software - Foreword (1144 words)

We wouldn't have to spend so much time, money, and effort on network security if we didn't have such bad software security.

Think about the most recent security vulnerability you've read about. Maybe it's a killer packet, which allows an attacker to crash some server by sending it a particular packet. Maybe it's one of the gazillions of buffer overflows, which allow an attacker to take control of a computer by sending it a particular malformed message. Maybe it's an encryption vulnerability, which allows an attacker to read an encrypted message, or fool an authentication system. These are all software issues.

Sometimes, network security can defend against these vulnerabilities. Sometimes the firewall can be set to block particular types of packets or messages, or to only allow connections from trusted sources. (Hopefully, the attacker is not already inside the firewall, or employed by one of those trusted sources.) Sometimes the intrusion detection system can be set to alarm if the particular vulnerability is exploited. Sometimes a Managed Security Monitoring service can catch the exploit in progress, and halt the intrusion in real time. But in all of these cases, the original fault lay with the software. It was bad software that resulted in the vulnerability in the first place.

Bad software is more common than you probably think. The average large software application ships with hundreds, if not thousands, of security- related vulnerabilities. Some of these are discovered over the years, as people deploy the applications. Vendors usually patch the vulnerabilities they learn about, and the hope is that users install the vendor -supplied patches (and that they actually work) or that the network security devices can be reconfigured to defend against the vulnerability. The rest of the software vulnerabilities remain undiscovered, possibly forever. But they're there. And they all have the potential to be discovered and exploited.

Bad software is to blame.

It's the software development system that causes bad software. Security is not something that can be bolted on at the end of a development process; it has to be designed in correctly from the beginning. Unfortunately, those who are in charge of a product's security are not in charge of the software development process. Those who call for increased security don't win against those who call for increased functionality. Those who champion principles of secure software design are not in charge of the software release schedule. The rallying cry of software companies is "more and faster," not "less, slower, and more secure."

In order to create secure software, developers need to understand how to design software securely. This understanding has several components. We need better education; programmers must learn how to build security into their software design, and how to write their code securely. We need better computer languages and development tools, tools that catch common security vulnerabi lities in the development process and make it easier to fix them. And most importantly, we need awareness: awareness of the risks, awareness of the problems, awareness of the fixes. This is not something that can happen overnight, but it is something that will have to happen eventually.

Building Secure Software is a critical tool in the understanding of secure software. Viega and McGraw have done an excellent job of laying out both the theory and practice of secure software design. Their book is useful, practical, understandable, and comprehensive. It won't magically turn you into a software security expert, but it will make you more sensitive to software security. And the more sensitive you are to the problem, the more likely you are to work toward a solution.

We're all counting on you, the readers—the programmers, the software architects, the software project managers—to work toward a solution. The software industry won't, because today there's no market incentive to produce secure software.

It's the lack of software liability. There is no market incentive to produce secure software because software manufacturers risk nothing when their products are insecure. The software industry has proven, product after product, that you can produce vulnerability-laden software and still gain market share. If automobile manufacturers were immune from product liability, I would be able to buy a motorcycle with a nitrous oxide injector spliced into my fuel line. I would be able to push the acceleration faster than the brakes could handle. I would be able to have any performance features that the manufacturers could build, regardless of the consequences. But I can't, because the motorcycle manufacturers would face lawsuits for marketing an unsafe product.

There are no similar lawsuits in software. Look at a shrink -wrap license from a piece of software you've just purchased (or even from one you've helped write). That software is sold "as is," with no guarantees of any kind. There's no guarantee that the softw are will function as advertised, let alone that it will be secure or reliable. There's no guarantee that it won't crash your entire network immediately upon installation. In fact, the software vendor is explicitly absolved of any liability if it does.

For years I have preached that computer security, like all security, needs to be thought of in terms of risk management. We cannot avoid the threats through some magical application of technologies and procedures; we need to manage the risks. In this way, cyberspace is no different from the real world. People have been trying to solve security problems for over 4000 years, and there are still no technologies that allow someone to avoid the threats of burglary, kidnapping, or murder. The best we can do is to manage the risks. Why should cyberspace be any different?

Building Secure Software takes this risk-management approach to security. But while my recent focus is on detection and response, this book focuses on prevention. Most importantly, it focuses on prevention where it should occur: during software design. The book begins by setting expectations properly. The first few chapters discuss the importance of treating software security as risk management, and introduce the idea of making technical tradeoffs with both security and the end goal (software that works) firmly in mind. Chapters five and six get to the heart of secure design, providing ten guiding principles for building secure software, and describing how to audit software. Then things get technical. The balance of the book is devoted to the bits and bytes of software security, with deep coverage of everything from the pernicious buffer overflow to problems developers face coping with firewalls.

Software security has a long way to go. We need not only to learn how to do it, we also need to realize that it is important *to* do it. The fact that you have this book in your hands is a step in the right direction. Read it, learn from it. And then put its lessons into practice.

We're all counting on you.

Bruce Schneier
http://www.counterpane.com

SOFTWARE RISK MANAGEMENT
FOR SECURITY

GARY MCGRAW PH.D.

cigital

COMPUTER SECURITY is taking on new importance as electronic commerce metamorphoses from hype to reality. Large and small businesses alike are reinventing themselves as e-commerce players. The implications for computer security practice are immense. When bits count as money, protecting bits becomes as important as any other aspect of running a successful business.

One essential element shared by every modern information system is the software that determines how the system behaves. Today's software problems lead to spectacular real world failures of many different kinds, including security problems, reliability problems, and safety problems. It is probably only a matter of time before software causes the demise of a large company.

What can we do to combat software bugs lying at the root of these problems, especially in light of the rush to embrace e-commerce and the intense pressure of Internet time? How can we avoid treating security as an add-on feature, when, like dependability, security is really a property of a complete system? This column discusses an approach to security analysis that we have applied successfully over the last several years at Cigital. Our approach is no magic bullet, but it offers a reasoned methodology that has proven to be useful in the trenches.

A View From 50,000 Feet

OUR METHODOLOGY, like many useful things, is a mix of art and engineering. The idea is straightforward: Design a system with security in mind, analyze the system in light of known and anticipated risks, rank the risks according to their severity, test to the risks, and cycle broken systems back through the design process.

The process outlined above has one essential underlying goal: avoiding the unfortunately pervasive penetrate-and-patch approach to computer security—that is, avoiding the problem of desperately trying to come up with a fix to a problem that is being actively exploited by attackers. In simple economic terms, finding and removing bugs in a software system before its release is orders of magnitude cheaper and more effective that trying to fix systems after release. The many problems inherent in the penetrate-and-patch approach have been discussed elsewhere, but the main concerns are:

- Patches are rarely applied by overworked system administrators who would rather not tweak their functioning systems;
- patches are rushed out under immense pressure from the market and often introduce new problems of their own; and
- patches are superficial solutions that do not get at the heart of software problems.

Designing a system for security, and testing the system extensively before release, presents a much better alternative.

Risk Assessment

THERE IS NO SUCH THING AS 100 PERCENT SECURITY. In fact, there is a fundamental tension inherent in today's technology between functionality (an essential property of any working system) and security (also essential in many cases). A common joke goes that the most secure computer in the world is one that has its disk wiped, is turned off, and is buried in a ten foot hole filled with concrete. Of course, a machine that secure also turns out to be useless. In the end, the security question boils down to how much risk a given enterprise is willing to take on in order to solve the problem at hand effectively. Security is really a question of risk management.

The key to an effective risk assessment is expert knowledge of security. Being able to recognize situations where common attacks can be applied is half the battle. The first step in any analysis is recognizing the risks. This step is most effectively applied to a system's specification. Once risks have been identified, the next step is ranking the risks in order of severity. Any such ranking is a context-sensitive undertaking that depends on the needs and goals of the system at hand. Some risks may not be worth mitigating, depending, for example, on how expensive carrying out a successful attack might be. Ranking risks is essential to allocating testing and analysis resources further down the line. Since resource allocation is a business problem, making good business decisions regarding such allocation requires sound data.

Given a ranked set of potential risks in a system, testing for security is possible. Testing requires a live system and is an empirical activity requiring close observation of the system under test. Security tests often do not result in clear-cut results like obvious system penetrations, though sometimes they do. More often a system will behave in a strange or curious fashion that tips off an analyst that something interesting is afoot. These sorts of hunches can be further explored.

Sound Software Engineering

ONE PREMISE OF OUR METHODOLOGY IS THE USE OF SOUND SOFTWARE ENGINEERING PRACTICES. Process is a good thing, in moderation. Any system that is designed according to well-understood requirements will be better than a system thrown together arbitrarily. An apt example of a security-related requirement is a requirement that states that some data must be protected against eavesdropping since it is particularly sensitive information.

From a set of requirements, a system specification can be created. The importance of solid system specification cannot be overemphasized. After all, without a specification, a system cannot be wrong, it can only be surprising! And when it comes to running a business, security surprises are not something we want.

A solid specification draws a coherent big-picture view of what the system does and why the system does it. Specifications should be as formal as possible, without becoming overly arcane. Formality is extremely powerful, but it too is no silver bullet. Remember that the essential raison d'etre for a specification is understanding. The clearer and easier to understand a specification is, the better the resulting system will be.

The Importance of External Analysis

NOBODY DESIGNS OR DEVELOPS SYSTEMS POORLY ON PURPOSE. Developers are a proud lot, and for the most part they work hard to create solid working systems. This is precisely why a security risk analysis team should not include anyone from the design and development team. One essential way in which security testing differs from standard testing is in the importance of preserving a completely independent view of the system, divorced from design influences. In general testing, one person can play dual roles; a design team testing expert to improve testability early on, and an independent tester later in the process. In security testing there is a much greater risk of tunnel vision.

Putting together an external team is important for two main reasons:

- To avoid tunnel vision. Designers and developers are often too close to their systems and are skeptical that their system may have flaws.
- To validate design document integrity. The requirements and specifications that the designers and developers use should be clear enough that an external team can completely understand the system.

Another reason warranting the use of external teams is the expertise issue. Being an excellent programmer and understanding security problems are not the same thing.

The good news is that an external team need not be made up of high-priced external experts. Often it is good enough to have a team from your own organization made up of security experts who were not involved in design decisions. The bad news is that security expertise seems to be a rare commodity these days. Determining whether or not to seek help outside of your organization will depend on what the system you are designing is meant

to do and what happens if it is broken by attackers. If you are betting your business on a piece of code, it had better not fail unexpectedly.

An experienced team of external analysts considers myriad scenarios during the course of an analysis. Examples of scenarios include decompilation risks in mobile code systems, eavesdropping attacks, playback attacks, and denial of service attacks. Testing is most effective when it is directed instead of random. The upshot is that scenarios can lead directly to very relevant security tests.

Security Guidelines

SECURITY IS RISK MANAGEMENT. The risks to be managed take on different levels of urgency and importance in different situations. For example, denial of service may not be of major concern for a client machine, but denial of service on a commercial Web server could be disastrous. Given the context-sensitive nature of risks, how can we compare and contrast different systems in terms of security?

Unfortunately, there is no golden metric. But we have found in practice that the use of a standardized set of security analysis guidelines is very useful. Our most successful client makes excellent use of security guidelines in their risk management and security group.

The most important feature of any set of guidelines is that they create a framework for consistency of analysis. Such a framework allows any number of systems to be compared and contrasted in interesting ways.

Guidelines consist of both an explanation of how to do a security analysis in general, and what kinds of risks to consider. No such list can be absolute or complete, but common criteria for analysis—such as the Department of Defense's Trusted Computing Systems Evaluation Criteria, commonly called the Orange Book—can be of help.

Much of today's software is developed incredibly quickly under immense market pressure. Internet time now rivals dog years in duration, approaching a 7:1 ratio with regular time. Often the first thing to go under pressure from the market is software quality (of any sort). Security is an afterthought at best, and is often forgotten.

A Common Mistake

Bolting security onto an existing system is simply a bad idea. Security is not a simple feature you can add to a system at any time. Security is like fault tolerance, a system-wide emergent property that requires much advance planning and careful design.

We have come across many real-world systems (designed for use over protected proprietary networks) that were being reworked for use over the Internet. In every one of these cases, Internet-specific risks caused the systems to lose all their security properties. Some people refer to this problem as an environment problem, where a system that is secure enough in one environment is completely insecure when placed in another. As the world becomes more interconnected via the Internet, the environment most machines find themselves in is at times less than friendly.

It is always better to design for security from scratch than to try to add security to an existing design. Reuse is an admirable goal, but the environment in which a system will be used is so integral to security that any change of environment is likely to cause all sorts of trouble—so much trouble that well-tested and well-understood things fall to pieces.

Security Testing Versus Functional Testing

Functional testing dynamically probes a system to determine whether the system does what it is supposed to do under normal circumstances. Security testing is different. Security testing probes a system in ways that an attacker might probe it, looking for weaknesses to exploit. In this sense, advanced testing methodologies like software fault injection can be used to probe security properties of systems. An active attack is an anomalous circumstance that not many designers consider.

Security testing is most effective when it is directed by system risks that are unearthed during a risk analysis. This implies that security testing is a fundamentally creative form of testing that is only as strong as the risk analysis it is based on. Security testing is by its nature bounded by identified risks (and the security expertise of the tester).

Code coverage has been shown to be a good metric for understanding how good a particular set of tests is at uncovering faults. It is always a good idea to use code coverage as a metric for measuring the effectiveness of functional testing. In terms of security testing, code coverage plays an even more critical role. Simply put, if there are areas of a program that have never been exercised during testing (either functional or security), these areas should be immediately suspect in terms of security. One obvious risk is that unexercised code will

include trojan-horse functionality whereby seemingly innocuous code carries out an attack. Less obvious (but more pervasive) is the risk that unexercised code has serious bugs that can be leveraged into a successful attack.

Dynamic security testing can help ensure that such risks don't come back to bite you. Static analysis is useful as well. Many of today's security problems are echoes of well-understood old problems that can come around again. The fact that 80 percent of 1998's CERT alerts involved buffer overflow problems emphasizes the point. There is no reason that any code today should be susceptible to buffer overflow problems, yet they remain the biggest source-code security risk today.

It is possible to scan security-critical source code for known problems, fixing any problems encountered. Current research is exploring the utility of static source-code scanning. The key to any such approach is a deep knowledge of potential problems.

There are many areas today in which software must behave itself. Good software assurance practices can help ensure that software behaves properly. Safety-critical and high assurance systems have always taken great pains to analyze and track software behavior. With software finding its way into every aspect of our lives, the importance of software assurance will only grow. We can avoid the band-aid-like penetrate-and-patch approach to security only by considering security as a crucial system property and not as a simple add-on feature.

Computer security is becoming more important because the world is becoming highly interconnected and the network is being used to carry out critical transactions. The environment that machines must survive in has changed radically. Deciding to connect a LAN to the Internet is a security-critical decision. The root of most security problems is software that fails in unexpected ways. Though software assurance has much maturing to do, it has much to offer to those practitioners interested in striking at the heart of security problems.

Gary McGraw, Ph.D. is Vice President of Corporate Technology at Cigital and co-author of the books Securing Java (1999) and Software Fault Injection (1998). Contact him at gem@cigital.com.